HYBRID SYSTEM DIAGNOSIS AND CONTROL RECONFIGURATION FOR MANUFACTURING SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Nicholas Chung Propes

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in Electrical and Computer Engineering

Georgia Institute of Technology
April 2004

HYBRID SYSTEM DIAGNOSIS AND CONTROL RECONFIGURATION FOR MANUFACTURING SYSTEMS

Approved by:

Dr. George Vachtsevanos, Advisor

Dr. Yorai Wardi

Dr. Magnus Egerstedt

Dr. Linda Wills

Dr. J. Lewis Dorrity

April 2, 2004

DEDICATION

For Mom the Artist, Dad the Mathematician,

Lars the Untouchable Crime-fighter, and Aimee the Architect

ACKNOWLEDGMENT

This thesis is dedicated to my parents and siblings who have always supported me throughout my academic endevors with love, support, and understanding.

The author wishes to express his deepest gradtitude to the Georgia Institute of Technology and the Manufacturing Research Center for the opportunity to learn. To my advisor, Dr. George Vachtsevanos (Dr. V), I cannot express my appreciation for his patience and enthusiasm throughout my academic career at Georgia Tech. He is a father figure to many of us far from home. I would also like to express my thanks to Dr. J. Lewis Dorrity for allowing me the opportunity to teach a few classes and work with students in the Industrial Controls Laboratory. I would like to thank the members of my committee, Dr. Yorai Wardi, Dr. Magnus Egerstedt, Dr. Linda Wills, and Dr. J. Lewis Dorrity for their time, support, and suggestions.

To my laboratory friends past and present, I enjoyed our collaboration on many different and exciting research topics ranging from chicken bone detection to prognostic enhanced diagnostic systems. Thank you also for teaching me a few words in Chinese, Korean, Spanish, etc.

To my machines that often seemed to repair themselves with just a magic touch of my hand, thanks for giving me a few maintenance headaches that enhanced my learning experience and confidence with machinery.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Designing models and controls for manufacturing processes in the presence of failures is a complex task that requires a high degree of fault tolerance for economic and factory safety reasons. This thesis presents a manufacturing oriented solution that focuses on how one may model and design reconfigurable controls for manufacturing systems in a hybrid framework that are subjected to various failures from the product to subsystem levels.

Modeling manufacturing systems is often a laborious task especially for processes that require many different operations and have several different types of subsystems. Object-oriented design offers many significant advantages that can aid the control designer in this task such as object reusability, simplified model modification, and organization. In addition, manufacturing processes are often modeled as event- or continuous-time driven models, but not under a hybrid framework that incorporates both types of dynamics. However, there are many interesting behaviors such as switching instability that can occur and cannot be described by these "pure" models. In essence, it is desirable to model manufacturing systems in a hybrid framework which fuses the event- and time-driven mechanics of the process together to provide a more descriptive model. This thesis provides a method one can model manufacturing processes in an object-oriented, hybrid systems framework utilizing simple Petri nets to describe the flow of events and differential equation models to describe the system dynamics.

Failures in all systems including manufacturing systems are inevitable. Detecting these failures correctly through some sensor network is essential to the reconfiguration

process and is known as diagnostics. Diagnostic-software processes analyze incoming sensor signals to classify the behavior into separate failure mode classes. Once the failure has been successfully classified, the controls are reconfigured in an attempt to bring the system back to useful operation. However, typical diagnostic software engines are computationally inefficient and error prone, because they do not include information about the operating mode. The operating mode can affect how failures might be detected such as what features need to be extracted and what diagnostic algorithms are to be utilized. When considering the wide variety of operating modes that a large manufacturing system undergoes, identifying the operating mode is a very important requirement for robust diagnostics performance. This thesis provides a mode identification technique that incorporates both measured events and time-driven dynamics.

The reconfigurable control problem for manufacturing systems can be split into the two cases of product quality and system performance. When product quality is reduced due to a failure in the system, it is desirable to reconfigure the controls so that the product quality returns within acceptable bounds. These bounds are usually described statistically through histograms derived from collected data over a number of runs. This thesis provides a way one may adjust the set-points of the system by comparing features of histograms in order to correct the deviations in product quality.

When a subsystem is affected by a failure, the subsystem may be rendered inoperable, however, there are cases where redundant actuators and sensors can be utilized to continue the operation of the process but in a less efficient manner. This thesis suggests a method to reconfigure the controls by rerouting products away from disabled

subsystems toward similar working subsystems, adjusting operation flow of the process, and utilization of a standard adaptive controller at the low-level.

CHAPTER 1

INTRODUCTION

1.1 Problem Statement/Motivation

Manufacturing processes are complex large scale systems that demand a high degree of autonomy in order to increase the amount of quality product produced while operating under uncertain environmental conditions. While unattended they are frequently subjected to fault modes that may degrade the product quality. Therefore, there is a need to automatically detect process anomalies, determine their impact on product quality and reconfigure process controls to maintain an acceptable product until the emergency is alleviated. The problem of control reconfiguration for manufacturing can be broken into three important operations: modeling, fault detection and identification, and control adjustment. There are many approaches relating to each of these operations, however, there lacks an object-oriented, hybrid modeling framework which can be utilized for solving a wide variety of reconfiguration problems relating to manufacturing processes such as rescheduling and product quality assessment and correction.

Fault detection and identification techniques, utilizing intelligent methods such as feature extraction, fuzzy logic, neural networks, etc., can be wasteful of computational resources by not taking into consideration the system operating mode. The operating mode should determine how and what features are extracted from sensor signals and what classifiers should be used. Moreover, the operating mode could also be used to determine usage patterns that would support the prediction capabilities of prognostic algorithms used in condition-based maintenance architectures. However, there lacks a mode

identification technique that takes both events and time-driven dynamics of a process to determine the operating mode.

1.2 Research Scope

This section defines some preliminary terminology about the problem of reconfiguration and then discusses the scope of this research. Hybrid systems are systems that are composed of event- and time-driven dynamics. Event-driven dynamics are those dynamics that propagate upon the initiation of an event. Such events can be initiated through the internal dynamics of the various systems, such as a robot picking up a part, or through the control signals such as a user pressing a stop button. Time-driven dynamics progress through the passage of time based upon the physical structure of the system. Figure 1.1 shows a manufacturing system under a hybrid system, modeling framework. Each subsystem is modeled as a separate hybrid system that interacts by events through a routing control. Product quality is assessed and set-points governing the time-driven dynamics are reconfigured if the product does not meet specifications. The purpose of this research is to develop an object-based modeling framework for hybrid systems which captures these essential behaviors. For purposes of reconfiguration, only a modeling framework for simple rerouting operations will be pursued, while the product quality reconfiguration control will be developed.

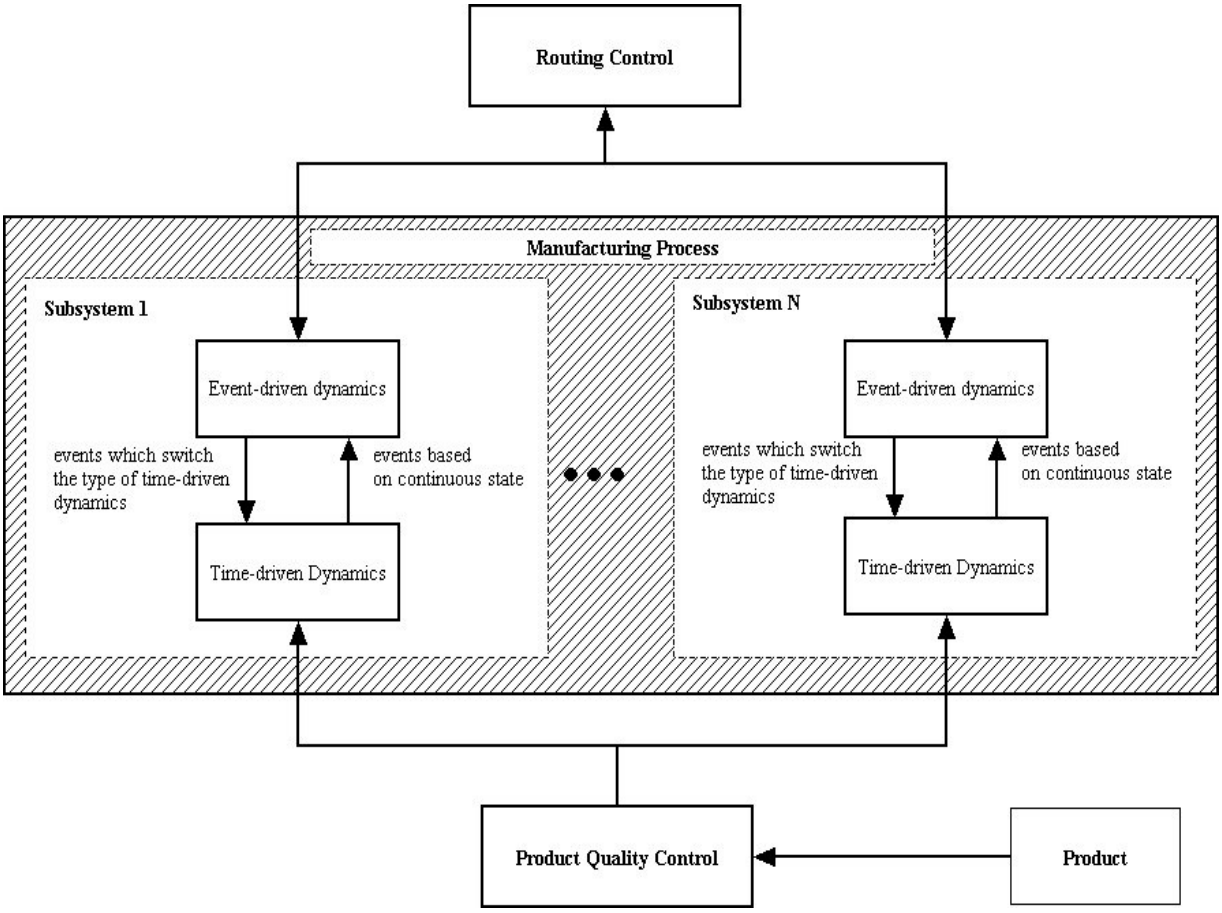

Figure 1.1. The reconfiguration architecture.

Mode identification is an operation upon measured variables that determines the current operating mode of the system. During operation, a manufacturing system may undergo various operating mode changes such as no motion, moving from point A to point B, etc. Figure 1.2 shows an example of the operating modes of a robot. Each operating mode will have a great influence on how fault detection and identification should be performed. In the approach presented here, events and evaluation of the dynamics through the continuous state are both utilized to determine the current operating mode of the system.

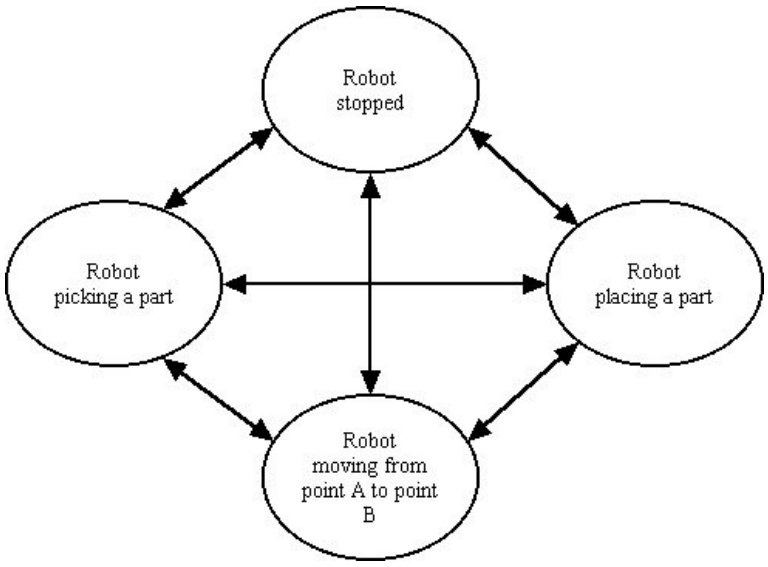

Figure 1.2. Example operating modes of a robot.

1.3 Assumptions

Assumptions made in the methodology present are:

- There are sensors available that can detect the product defects and that provide information necessary to determine a corrective course of action on the set-points.
- The underlying cause of the defects is due to environmental factors and not due to the problems in the low-level controllers.
- The process can wait while defects are created until data is available to construct accurate histograms for corrective action.
- Defects are not catastrophic and reconfiguration is possible.
- The operating mode can be determined through sensor measurements.

1.4 Thesis Organization

The thesis is organized as follows: Chapter 2 introduces some background on the subjects of hybrid systems, mode identification, and reconfigurable control. Chapter 3 describes in detail the object-based hybrid model, the mode identification architecture, and the control reconfiguration methodology for system and product levels. Chapter 4 gives the results of the methodologies presented in Chapter 3. Chapter 5 then concludes with a summary and the contributions of this research.

CHAPTER 2

BACKGROUND

2.1 Hybrid System Modeling

In the past, manufacturing systems were modeled either as purely event-driven or purely time-driven models. Petri nets, automata, language theory models, etc. were used to represent the event-driven dynamics of a manufacturing system, while differential or difference equations represented the time-driven dynamics. The theoretical research results of these "pure" models is extensive and made many control problem solutions tractable. However, systems are becoming more complex and methods to analyze and synthesize systems with both time- and event-driven dynamics in a systematic fashion are becoming more popular.

Current research has turned towards finding models that intertwine both time- and event-driven dynamics. These models are called hybrid system models. A two level hierarchy representation of a hybrid system is shown in Figure 2.1 [36]. The discrete state (i.e. the operational mode) determines the controllers, set-points, or representative dynamics of the time-driven portion of the system through the digital-to-analog (DA) interface. Events originating from measured time-driven signals determine mode transitions via the analog-to-digital (AD) interface.

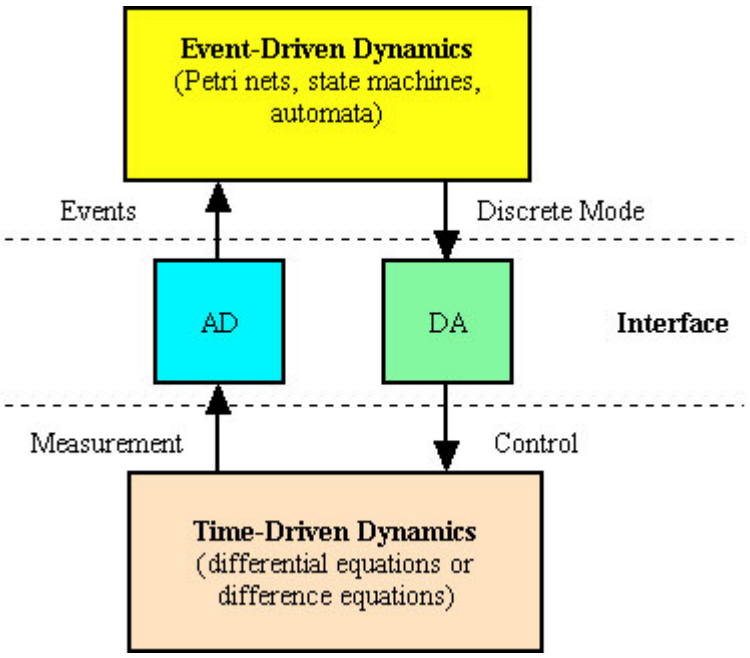

Figure 2.1. A hybrid system.

According to Branicky [13], there are four hybrid modeling approaches in the literature: Aggregation, Continutation, Automatization, and Systemization. Aggregation suppresses the continuous dynamics so that the hybrid system is purely event-driven. For example, Koutsoukos and Antsaklis [36] discretize the continuous state spaces of hybrid systems producing a discrete-event model. Continuation suppresses the event-driven dynamics so that the hybrid system is represented by a differential equation. Branicky has taken this approach [7]. Automatization views a hybrid system as network of interacting automata with the focus on input-output behavior. Systemization views a hybrid system as a network of interacting dynamical systems where the focus is on the state-space. Branicky has developed this approach in many papers [7-13].

Branicky introduces a hybrid system model called the general controlled hybrid dynamical system (GCHDS) [7]. A GCHDS is a 7-tuple,

$$H_c = [Q, \Sigma, A, G, V, C, F],$$

where,

- Q is the set of discrete states
- $\Sigma = \{\Sigma_q\}_{q \in Q}$ is a collection of controlled dynamical systems where *each* $\Sigma_q = [X_q, \Gamma_q, f_q, U_q]$ is a controlled dynamical system. X_q are continuous state spaces, f_q are the continuous dynamics, and U_q is the set of continuous controls.
- $A = \{A_q\}_{q \in Q}$, $A_q \subset X_q$ for each $q \in Q$, is the collection of autonomous jump sets.
- $G = \{G_q\}_{q \in Q}$, where G_q: $A_q \times V_q \rightarrow S$ is the autonomous jump transition map, parameterized by the transition control set V_q.
- $V = \{V_q\}_{q \in Q}$, represent the discrete dynamics and controls.
- $C = \{V_q\}_{q \in Q}$, $C_q \subset X_q$, is collection of controlled jump sets.
- $F = \{V_q\}_{q \in Q}$, where F_q: $C_q \rightarrow 2^S$ is the collection of controlled jump destination maps.

$S = \bigcup_{q \in Q} X_q \times \{q\}$ represents the hybrid state space.

This hybrid system model can be viewed as an automation whose arcs (G_q) have conditions that must be met for jumping to take place between vector spaces and/or states. Figure 2.2 shows a diagram of different vector fields, and how a state could possibly change due to the dynamics of a GCHDS. The D_q sets shown are jump destination states (i.e. sets where the state can jump into when changing vector fields). When the state hits the boundaries of a region, A_q, then G_q will determine the next state and vector field the system will enter. Also, when the state enters a region, C_q, then the state and vector field may change to any point in F_q dependant on the current state of the controls.

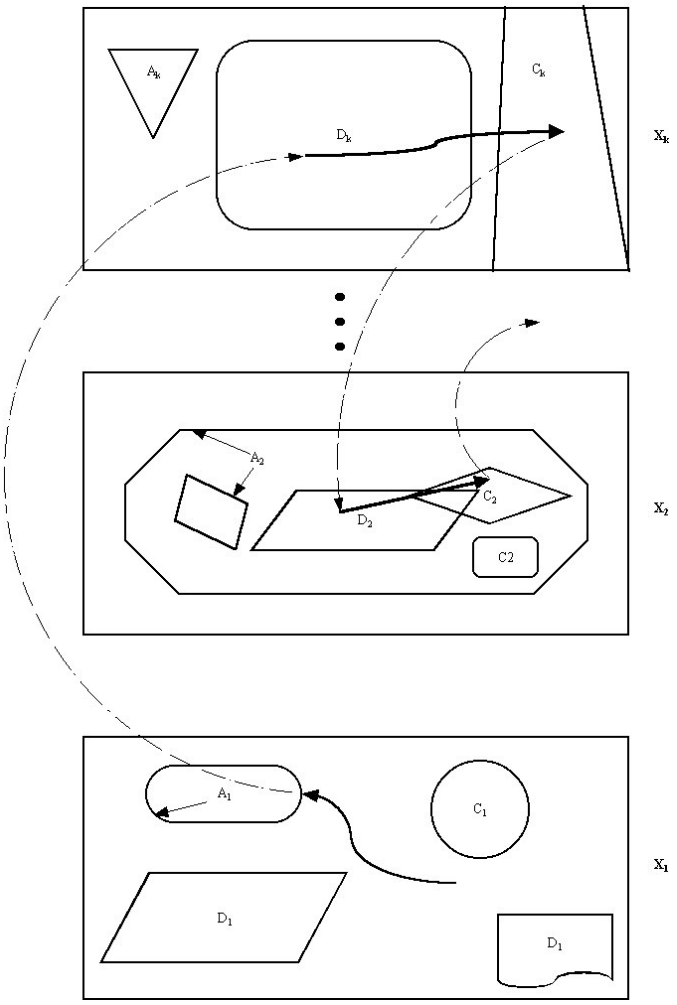

Figure 2.2. An example of GCHDS dynamics.

Some hybrid models are presented in the technical literature to solve application specific control problems. Cassandras et al present a simple manufacturing system model consisting of max-plus and differential equations in a classical optimization framework [15-17]. Non-cooperative aircraft control [52] and automated highway vehicles [38] use hybrid automata as models in a game theoretic framework. Hybrid Petri nets have been introduced in [2] for electronic component manufacturing systems.

Object-oriented design dissects a complex system into small understandable objects that interact. Carpanzano [14] et al introduce an object-oriented framework for representing hybrid systems, however, their "object" is more similar to a "module" as found in computer science literature. Al-Hasan extends Rumbaugh's Object-modeling Technique (OMT) for systems other than software modules in his dissertation [1]. However, the dynamics do not represent a truly hybrid system. It would be greatly beneficial to incorporate hybrid system modeling techniques into an OMT framework.

2.2 Mode Identification

The discrete state, Q, in the CGHDS model of Section 2.1 is sometimes referred to as the operating mode. The operating mode represents a specific type of system dynamics arising from controller switching, set-point modifications, or system dynamics changes. Mode identification (also known as mode discernability) addresses the problem of determining the operating mode of a system through processing of incoming sensor data. Modes of a hybrid system switch through the occurrence of an event. These events are triggered when controller input changes are applied to the system and/or when the state of the system moves into certain regions of the state space. For example, each line of code of a mill computer numeric control (CNC) program could represent the operational mode (i.e. cutting a curved surface, waiting for a robot signal, moving to home position, etc.) Figure 2.3 shows a mode diagram where circles are modes and arcs represent event conditions that cause transitions.

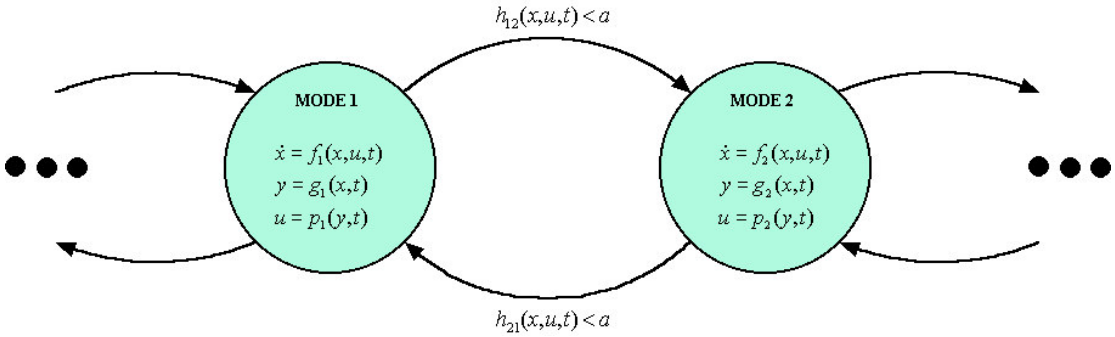

Figure 2.3. A mode diagram.

Koutsoukos et al [37] describe a mode identification framework for a laser printer. They assume that the printer is running in a sensor rich environment. It contains sensors that relay discrete signals, which, in turn, trigger controls while audio and current sensors are used to determine the mode. Measured signals are assumed to be a linear superposition of template signals:

$$y_i(t) = \sum_{j=1}^{n} \alpha_{ij} s_j(t - \tau_{ij}), i = 1,...,l \tag{2.1}$$

where,

- y_i is the i^{th} measured signal
- s_j is the j^{th} template signal for mode identification
- τ_{ij} is the onset of the template, s_j, for the i^{th} measured signal
- α_{ij} the sensor gain of template, s_j, for the i^{th} measured signal

Using the coefficients α_{ij} and τ_{ij}, conditional probability distributions can be constructed to estimate the new mode based upon measured signals. This type of framework ignores the event information from the sensors that provide the discrete signals. Moreover, the continuous signals themselves are not directly associated with the

dynamics of the machine (i.e. audio sensors do not measure the velocity of a drum directly). A methodology that incorporates both event information and direct, sensor readings would provide a more accurate determination of the operating mode.

In [4], Babaali and Egerstedt examine the observability properties of deterministic, discrete-time, switched, linear systems. Balluchi et al [5] investigate generic final-state asymptotically determinable, linear hybrid systems and show that these systems can be verified even if each of the time-driven expressions are not observable. Bemporad et al [6] show that observability and controllability properties cannot be easily deduced from component linear subsystems. Other observability results for hybrid systems can be found in [30-31][34].

2.3 Reconfigurable Controls

Recently, the design of reconfigurable controls for hybrid systems has been addressed in the technical literature. Figure 2.4 shows a typical reconfiguration framework consisting of a plant, the plant's low-level controller, a supervisory controller, a fault detection and identification module (FDI), and a controller reconfiguration module. The FDI module detects any failures in the system from measured plant signals and then, the reconfiguration module appropriately adjusts control signals to allow continued operation of the plant.

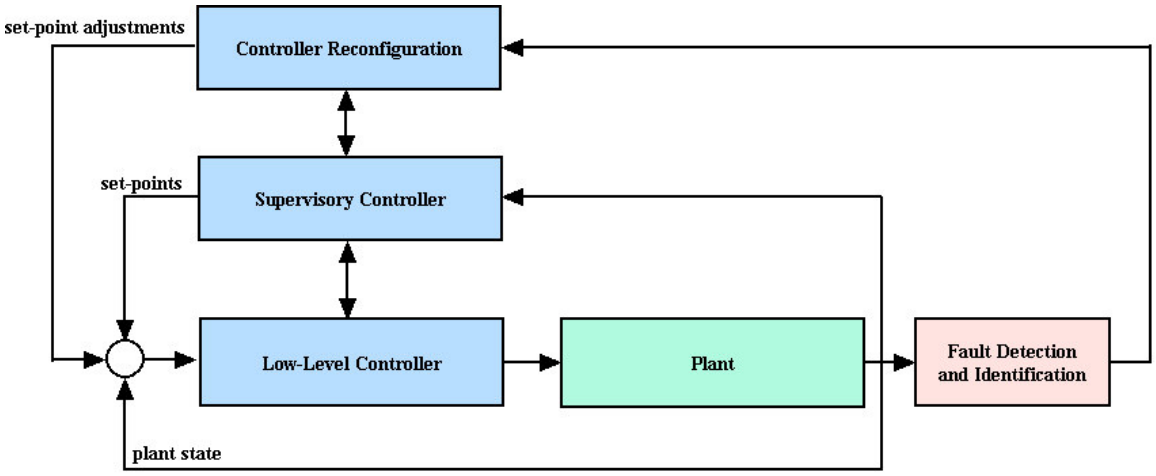

Figure 2.4. The reconfiguration framework.

Two major reconfigurable control paths have been pursued in control research. One way is to design a single control algorithm that is robust enough to handle failures and still achieve a certain level of performance. These controllers are known as passive controllers and one such example is the control mixer. They modify the transfer function of a faulty system so that it resembles closely the nominal transfer function. Zhenyu [60] constructs a control mixer directly from frequency domain information. Several control mixer designs are found by using the pseudo inverse solution and by doing such, stability can no longer be guaranteed. Yang [57] addresses the problem of stability using control mixers in an H-∞ framework. In manufacturing systems, these types of control methodologies are well suited for the designs of low-level controllers while operating in a single mode or modes that have similar characteristics.

The other reconfigurable control approach is classified as an active one where control strategies are switched when failures occur. Many of these methodologies address the reconfiguration issue through a low-level controller. Parasini et al [44]

develop a reconfiguration strategy for manufacturing systems that switches low-level controller gains upon the occurrence of a failure. A reconfiguration strategy where constraints upon control variables are altered when failures are present can be found in [23]. In his dissertation, Ramani [49] develops a methodology that selects between a number of active controllers when failures are detected and identified. Mahmood [40] utilizes a fuzzy Petri net to determine route choices in the presence of failures.

Cassandras et al [15-17] addresses the problem of finding optimal controls for subsystems with the arrival times of jobs known apriori. In their work, job arrival and departure times are events modeled through a max-plus equation and the subsystems themselves are modeled as differential equations. The job departure times depend upon the subsystem controls applied while a job is being processed.

In [11], Branicky shows that a hybrid system can become unstable even when switched between stable systems. He also gives conditions where multiple Lyapunov functions can be used to determine the stability of hybrid systems, namely, if there exist candidate Lyapunov-like functions for each vector field and that they obey a non-increasing condition along the state trajectory for a specific switching sequence, then the system is stable.

This thesis extends these ideas by:

- Representing the subsystems as hybrid systems through an object-oriented framework using Petri nets to model event-driven behavior.
- Allowing subsystem failures and introducing a reconfiguration strategy.
- Applying set-point adjustments to subsystems when the product quality is degraded.

CHAPTER 3

TECHNICAL APPROACH

3.1 The Object-Based Hybrid Modeling Framework

This section focuses upon an object-based hybrid modeling framework for manufacturing systems. An object-based approach will allow one breakdown a manufacturing system into objects representing well-understood subsystems as simple objects.

Sections 3.1.1-3.1.5 of this chapter describes a modeling framework developed in Al-Hasan's dissertation called Object-based World Modeling (OWM). It is composed of three submodels: object, dynamic, and functional. Section 3.1.5 introduces a modified version of the OWM dynamic submodel that will describe the hybrid dynamics utilizing a Petri net and a hybrid automaton.

3.1.1 Object-based World Model

Rumbaugh's Object Modeling Technique (OMT) is an object-oriented approach for software development. It provides a systematic way to model and design specific applications before writing code. Al-Hasan describes in his dissertation a modified version of OMT called Object-based World Modeling [1]. This approach was used to model the world environment for autonomous vehicle mission planning (i.e. for systems instead of software). A system is divided into objects representing system components. Submodels related to these objects describe the dynamics and operational dependencies within and external to the objects. The benefits of this methodology are that it captures the system's static structure, dynamics, and functionality in a unified framework. The

system model is divided into three submodels: the object submodel, the dynamic submodel, and the functional submodel as shown in Figure 3.1.

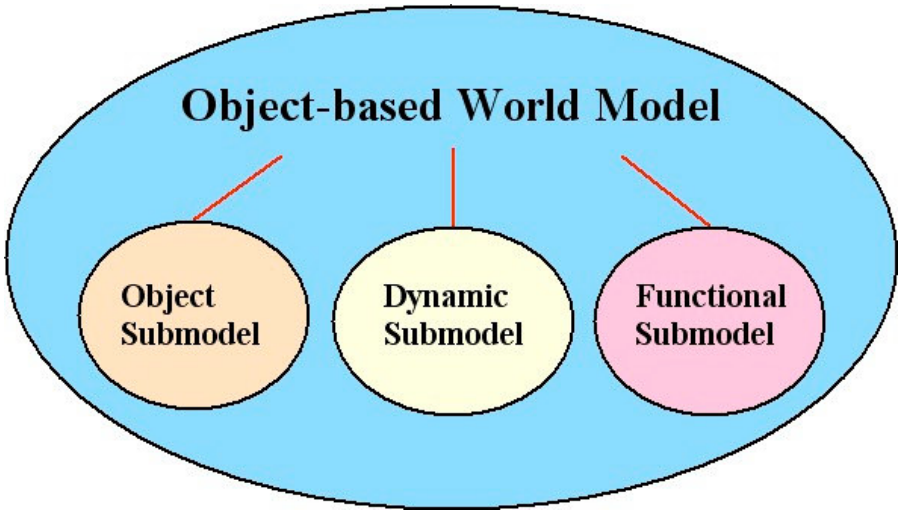

Figure 3.1. The Object-based World Model components.

3.1.2 The Object Submodel

The object submodel is the most important part of the modeling process. It represents the static structure of systems. The system is decomposed into objects that are connected by links. Figure 3.2 demonstrates an object submodel for a manufacturing system composed of a conveyor and a part. The conveyor object is broken down into motor and track objects. The links demonstrate both physical and operation connections between all objects. For example, the conveyor motor moves the track upon the initiation of the "move_track" operation, and the part supported by the track is moved by the "move_part" operation. There are four main elements of an object model: attributes, operations, operators, and links.

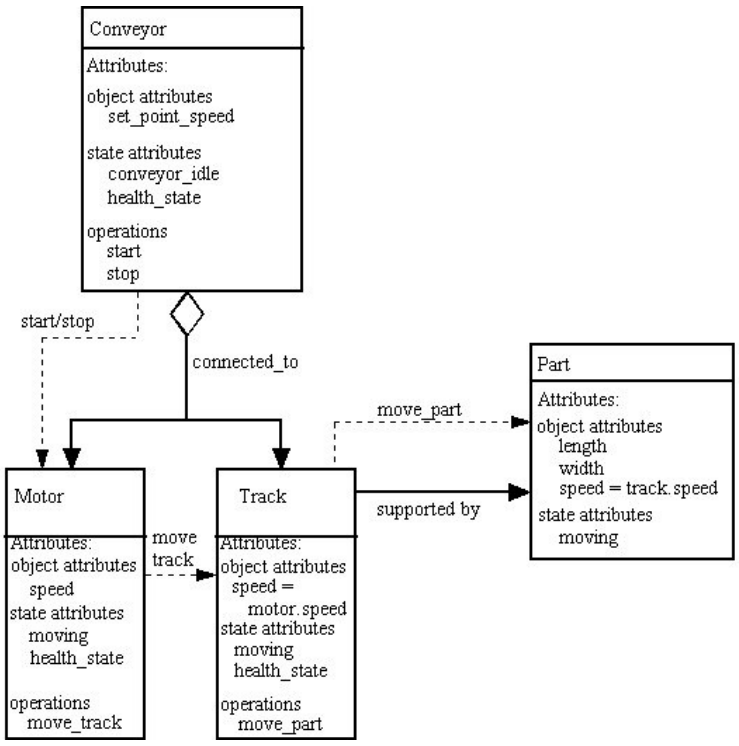

Figure 3.2. An object submodel.

The attributes of an object are divided into two different types: object and state attributes. An object attribute represents properties of the object such as length, color, speed, acceleration, etc. These can be considered as a detailed measurement of the object's features or static properties. The state attributes represent discrete states of an object such as moving, on/off, idle, healthy, inspected, etc.

Operations are tasks that an object can perform upon itself or other objects. There are two classes of operations, intransitive and transitive. Intransitive operations are those that are performed locally on the object such as move, compute velocity, etc. Transitive operations are operations that affect other objects. They involve operations such as move part A onto part B, pick part A off conveyor, etc. Operators describe operations in

template form. An operator template contains information for operations about target objects, preconditions in terms of states, effects upon other objects, time constraints, etc.

Links are used to describe physical or operation connections between objects in an object diagram. Physical links are depicted as solid line arrows that show physical dependence of objects. These types of links can be described as connected to, supported by, and being a component of. They are important because they relate object attributes such as position, velocity, etc. directly to other objects (e.g. a link of a robot arm will effect the velocity and speed of an end-effector). Operational links are shown as dashed line arrows that describe the flow of transitive operations. The head of the arrow depicts the target that the operation affects. The tail of the arrow determines what object initiates the operation.

3.1.3 The Dynamic Submodel

Each object can have a state diagram associated with it termed a dynamic submodel. This state diagram describes how state attributes change due to conditions on object attributes and operations (see Figure 3.3). The state diagram contains states represented as ovals, directed arcs which represent how states change due to events, and a ball-tailed arrow depicting the initial state. Each state represents the discrete modes or mode state attributes will take i.e. moving, stopped, etc. Events cause the transitions between states to occur. Events can come in the form of a Boolean function on the object attributes, operations or operators, or external events. The state diagram, as described by Al-Hasan, is a simple automaton with no description of time-driven dynamics.

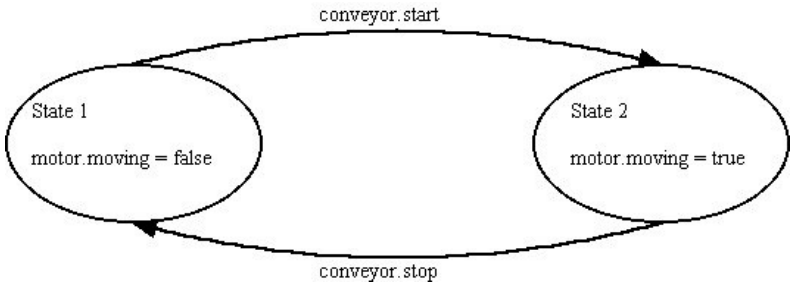

Figure 3.3. A state diagram.

3.1.4 The Functional Submodel

The functional submodel directs the information flow between object operations. Before an object operation can be initiated, information from other object operations may be needed. The functional submodel displays this operational dependence in an information flow diagram (see Figure 3.4). This diagram has ovals that represent operations and directed arcs that represent the priorities of operations. For example, before an inspection computer program ("inspection_pc") can compute features ("inspection_pc.convert_to_features") in Figure 3.4, the operations "temperature_sensor.measure" and "camera.take_image" must be completed. External operations (i.e. operations initiated from other objects) are located outside a dotted box and are enclosed by a dotted oval. In some cases, labels are attached to the arcs describing the frequency (e.g. 5 Hz) an operation will request information from other operations. Thus, there is usually a clock object that is used to handle the timing of these operations.

Updating the object attributes is also an issue. There are several ways one can update the object attributes. However, in this proposal, it will be assumed that all sensor

data is gathered based upon some multiple of a global clock frequency which is similar to how a PLC operates (i.e. in a cycle).

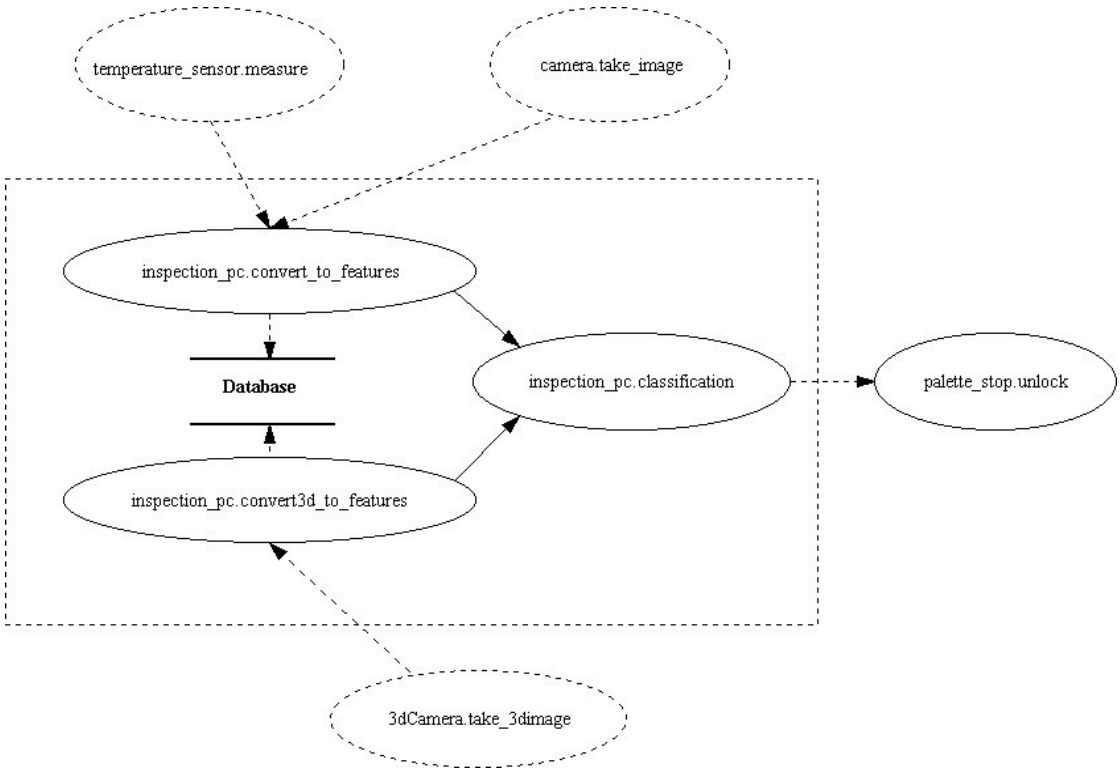

Figure 3.4. An information flow diagram.

3.1.5 The Object-based Hybrid Dynamic Submodel

A manufacturing system modeling method, called an Object-based Hybrid Model (OHM), is described in this section. It is based upon Al-Hasan's Object-based World Model. The difference between OHM and OWM lies with the dynamic submodel. The dynamic submodels will be composed of the main elements, as shown in Figure 3.5. The supervisory Petri net sits at the top level and its purpose is to coordinate subsystem start/stop events. At the mid-level, mode changes, in terms of the chosen controllers, set-points, and system parameters, will take place based upon events from other Petri nets

describing the subsystem. The low-level, time-driven subsystems will utilize differential equations to provide the time-driven dynamics.

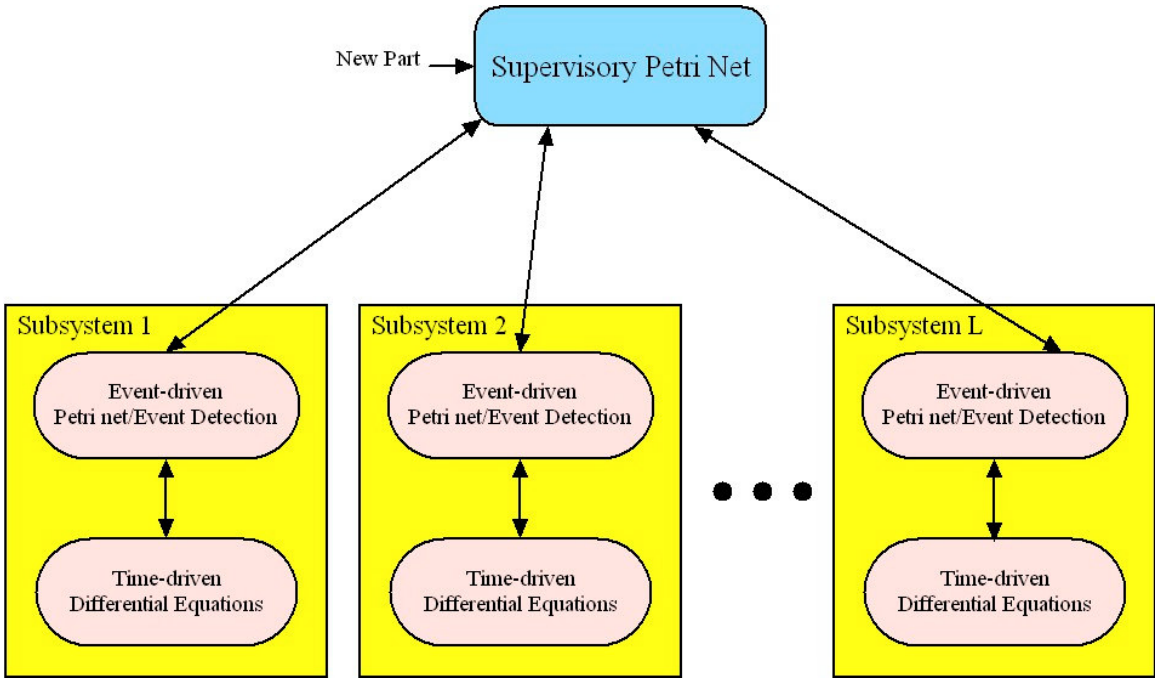

Figure 3.5. The Object-based hybrid model architecture.

The Object-based Hybrid Dynamic Submodel is defined by a 6-tuple,

$$O = (S, G, \Sigma, \gamma, \alpha, \beta)$$

where,

- $S = (P^S, T^S, D^{S+}, D^{S-}, E^S, \mu^S, \mu_0^S)$ is the supervisory Petri net where, $P^S = \{p_1, p_2, ..., p_n\}$ is a set of places, $T^S = \{t_1, t_2, ..., t_m\}$ is a set of transitions, D^{S+} is a mapping $P^S \times T^S \to \{0,1\}$ describing the existence of arcs from transitions to places, D^{S-} is a mapping $P^S \times T^S \to \{0,1\}$ describing the existence of arcs from places to transitions, $E^S \in \{0,1\}^{m \times m}$ is an event mask, diagonal matrix that prevents or

enables certain transitions from firing, $\mu^S \in Z^n$ is a vector describing the current state of the Petri net, $\mu_0^S \in Z^n$ is a vector describing the initial state of the Petri net.

- $G = (P_i, T_i, D_i^+, D_i^-, E_i, \mu_i, \mu_{i0})$ is a Petri net describing the operation flow of subsystem $i \in \{1,2,...,L\}$ due to events where, $P_i = \{p_1, p_2, ..., p_{n_i}\}$ is a set of places, $T_i = \{t_1, t_2, ..., t_{m_i}\}$ is a set of transitions, D_i^+ is a mapping $P_i \times T_i \to \{0,1\}$ describing the existence of arcs from transitions to places, D_i^- is a mapping $P_i \times T_i \to \{0,1\}$ describing the existence of arcs from places to transitions, $E_i \in \{0,1\}^{m_i \times m_i}$ is an event mask, diagonal matrix with elements 0 or 1 that prevents or enables certain transitions from firing, $\mu_i \in Z^{n_i}$ is a vector describing the current state of the Petri net, $\mu_{i0} \in Z^{n_i}$ is a vector describing the initial state of the Petri net.
- $\Sigma = (\Sigma_{q_i})_{q_i \in \{1,2,...,n_i\}}$ is a collection of controlled dynamical systems where each $\Sigma_{q_i} = (X_{q_i}, \Gamma_{q_i}, f_{q_i}, U_{q_i})$ is a controlled dynamical system, X_{q_i} are continuous state spaces, f_{q_i} are the continuous dynamics, and U_{q_i} is the set of continuous controls. This means each Petri net (G) place of subsystem, i, has a controlled dynamical system associated with it.
- $\gamma \in Z^L$ is the failure index vector where element γ_i, $i \in \{1,2,...,L\}$ of γ is the failure index for subsystem i. This index identifies the type of failure that occurs in the various subsystems.

- $\alpha = (\alpha_i)_{i\in\{1,2,\ldots,L\}}$ is a collection of mappings from the supervisory state, μ^S, the state of the system, X_{q_i}, and γ to subsystem i's, Petri net event mask, E_i (i.e. $\alpha_i(\mu^S, X_q, \gamma) \rightarrow E_i$).
- β is a mapping from the subsystem discrete states, $\mu_{i\in\{1,2,\ldots,L\}}$, and γ to the supervisory Petri net's event mask, E^S (i.e. $\beta(\mu_{i\in\{1,2,\ldots,L\}}, \gamma) \rightarrow E^S$).

Figure 3.6 represents the OHM dynamic submodel. The supervisory Petri net interacts with the subsystems through the state of the supervisory Petri net state and the mapping, β. The state of the supervisory Petri net tells which subsystem is operational. The event mask, E^S, prevents the supervisory Petri net from firing certain transitions while subsystems are performing actions on a part in the manufacturing system. The submodel Petri net and dynamical systems interact through the state of the submodel Petri net which selects the current dynamical system (i.e. operation) that is currently operating and the mapping, α_i, that selects the event mask, E_i. This event mask prevents the submodel Petri net from firing certain transitions until an operation is complete. The failure index vector modulates the mappings, α and β so that certain events cannot be fired while under a failure condition.

The evolution of the supervisory and submodel Petri nets are described by the equations:

$$\mu^S(k+1) = \mu^S(k) + D^S E^S t^S(k) \tag{3.1}$$

$$\mu_i(k+1) = \mu_i(k) + D_i E_i t_i(k) \tag{3.2}$$

where,

$$D^S = D^{S+} - D^{S-} \tag{3.3}$$

$$D_i = D_i^+ - D_i^- \tag{3.4}$$

and $t^S \in Z^m$ selects the supervisory Petri net transition to be fired and $t_i \in Z^m$ selects the subsystem Petri net transition to be fired. Both vectors contain 0's or 1's as elements.

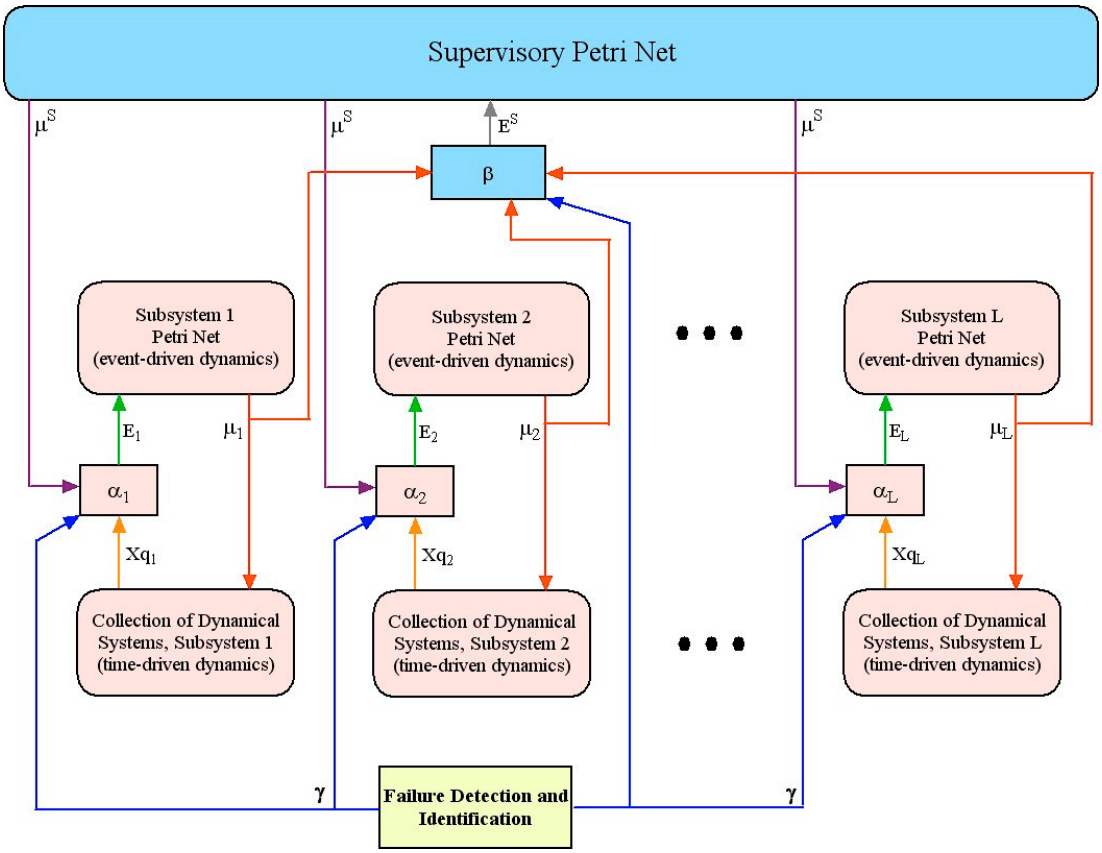

Figure 3.6. The OHM dynamic submodel.

For a subsystem divided into objects, Figure 3.7 shows that it has a similar structure however, the Petri net state for each object is combined and relayed to the supervisory and objects event mask. In addition, appropriate state variables are also distributed to all objects. This offers some strong flexibility in the dynamics since each object can switch modes separately.

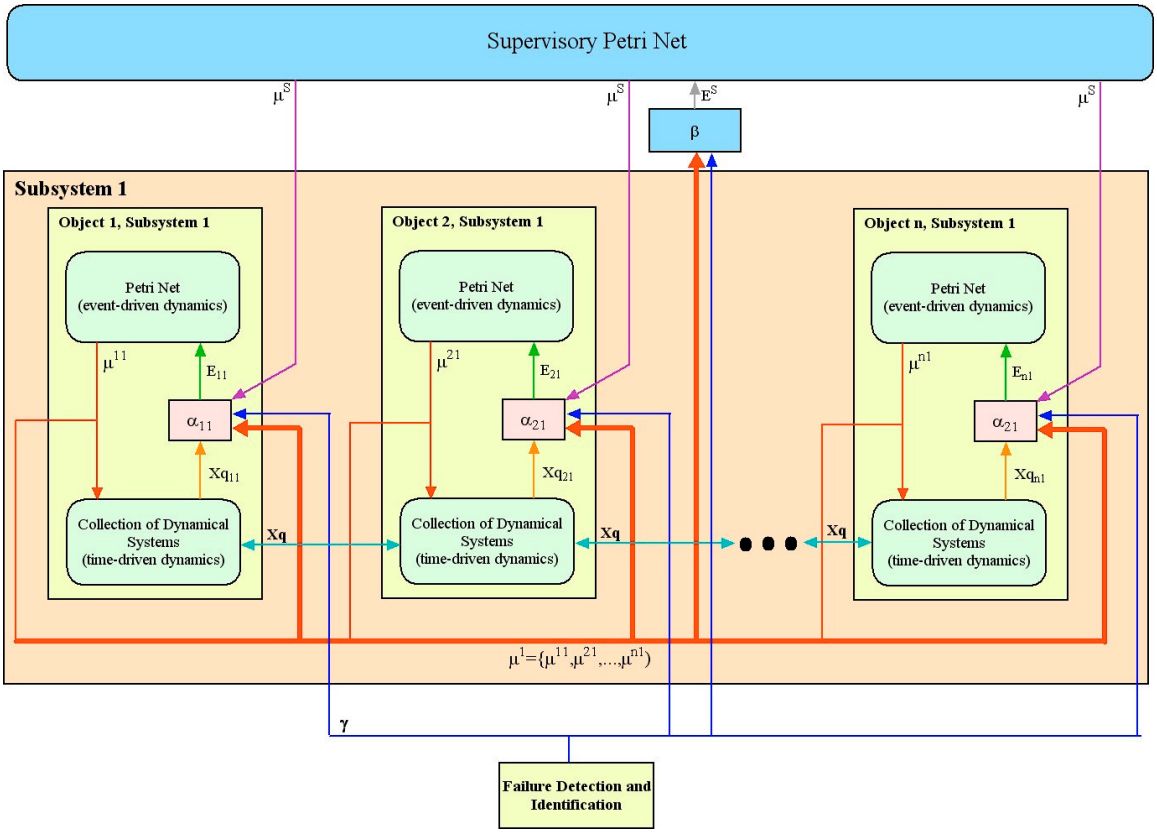

Figure 3.7. The OHM architecture with objects for a single subsystem.

The simulation of the dynamic submodel follows the steps below:

1. Initialize all Petri net states.
2. Initialize all dynamical system states.
3. Evaluate event mask matrices through α's and β.
4. Update subsystem Petri net states.
5. Perform dynamical subsystem simulation.
6. Update supervisory Petri net state.
7. Go to step 3 or end when there are no more parts to be processed.

The benefits of OHM are the following:

- It is a general hybrid model framework for many manufacturing systems.
- It is a modular framework facilitating easy exchange of new manufacturing subsystems.
- Mode transition events and high-level operations can be represented.
- Petri net theory can be applied to the event-driven part of the model.
- If the subsystem Petri net places are considered operations, then it includes the functional submodel.
- Failures can be modeled through disabled transitions.

3.2 Mode Identification

Mode identification is the process of identifying the current operating mode of the system through examination of the sensor measurements. The mode identification architecture is composed of three separate modules, a fuzzy logic classifier, a fuzzy Petri net, and an evidence combiner module as shown in Figure 3.8. The fuzzy logic classifier is used to determine the mode by examining the features describing the dynamics of the process. The fuzzy Petri net is used to determine the mode by detecting events. The evidence combiner combines the results of the fuzzy classifier and fuzzy Petri net and then outputs the identified mode. Therefore, this mode identification combines both events and time-driven dynamics information to determine the current operational mode of the system. This section describes all three of these modules in detail.

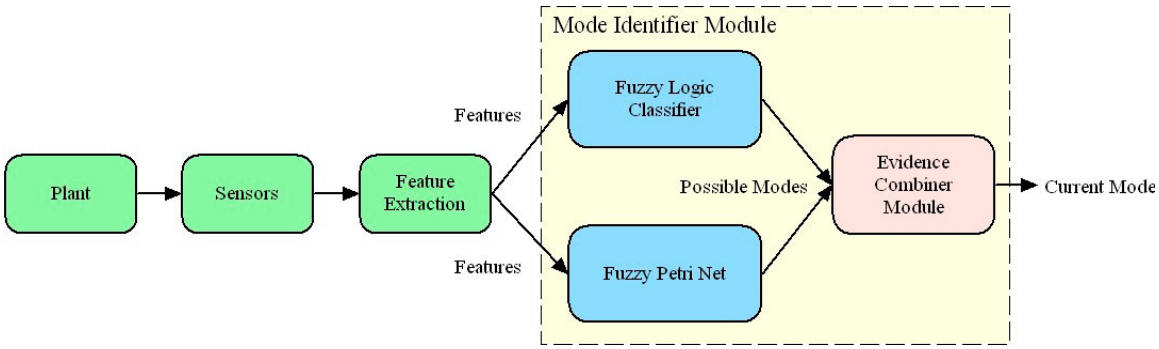

Figure 3.8. The mode identification architecture.

3.2.1 Fuzzy Petri Net as a Mode Identifier

The fuzzy Petri net has the ability to incorporate uncertainty in the detection of an event and a structure representing the relations between modes and events. Events have a degree of uncertainty associated with them and many can be expressed as if-then rules. Each mode will have a possible set of transitioning modes described by the net structure thus reducing the search space.

Events can occur from external inputs through a user, autonomously through control signals and system state variables, failures, etc. In many hybrid system frameworks, events occur when a function on the state and control variables crosses a threshold value that causes a transition from the current mode to another. Examples of event definitions are “the temperature is decreasing slowly below 80 degrees”, “the part arrived at the robot station”, “the mill has started cutting”, “the conveyor has blown a fuse”, etc.

The fuzzy Petri net is used to determine the current operating mode by detecting events signifying a switch in operating mode. The fuzzy Petri net used in this module consists of two parts, the structure of the net and the fuzzy membership function associated with each transition.

The fuzzy Petri net structure is composed of places that represent the operating mode, transitions which represent events, and arcs which describe how modes change due to events. A simple structure is shown in Figure 3.9.

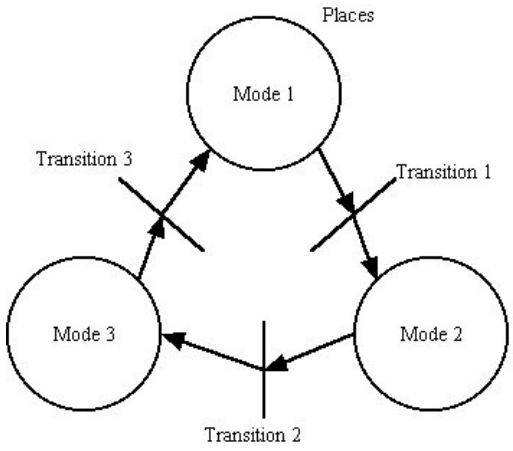

Figure 3.9. A Petri net structure.

The modes in the figure above describe 3 operating modes and 3 transitions that determine how modes are related to events. For illustration, if the past mode was Mode 1, then the only possible transition is to Mode 2 (i.e. if transition 1 fires). Two matrices are used to describe the arcs from places to transitions and arcs from transitions to places. The Input Matrix describes arcs from places to transitions and is of dimension, # of places by # of transitions.

$$InputMatrix = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The Output Matrix describes arcs from transitions to places and is of the dimension, # of places by # of transitions. For the example above this matrix would be:

$$OutputMatrix = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.6}$$

To determine the possible mode transitions when the past mode is place n, take the nth row of the input matrix and note the columns in that row that are 1's. Then, the rows of the output matrix with 1's corresponding to the columns found in the previous step yield the possible modes.

Each transition also has at least one fuzzy membership function describing the possibility that the event has been fired due to some input signal (e.g. features). There is also a threshold on the possibility that determines whether or not the event has fired. If the resulting possibility is greater than or equal to the threshold, then the transition has fired. It also is important to know which side the input signal comes from towards the membership function as shown in Figure 3.10. Here we assume that the feature evolution is monotonic.

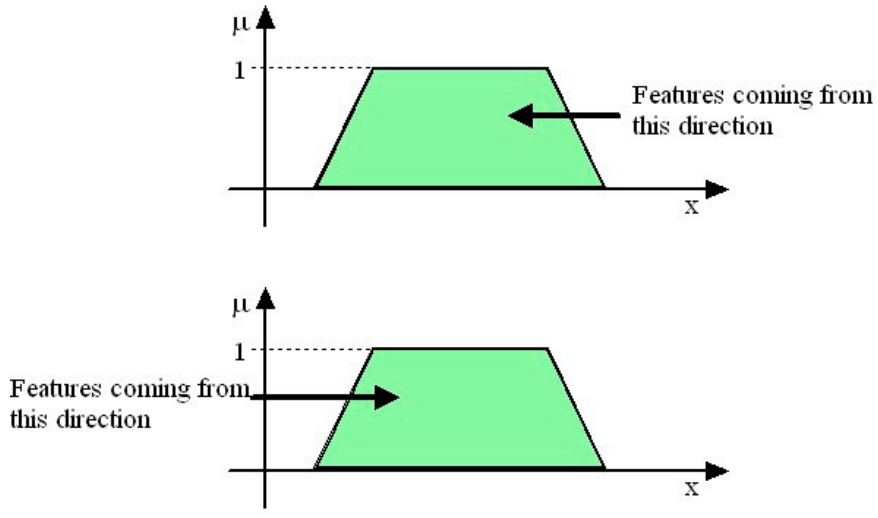

Figure 3.10. The direction of features corresponding to the membership function.

To determine the possibility, it is required that the possibility before and after the condition is examined. The possibility before the condition is calculated by extending the membership function to the max value in the opposite direction of the signal direction and then evaluating this modified membership function as in Figure 3.11. This can be described by the equation:

$$P_{before}(a) = \sup_{x \geq a}(\mu(x)) \tag{3.7}$$

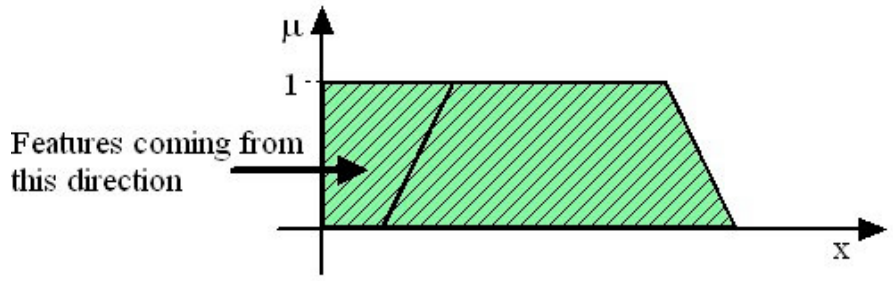

Figure 3.11. Possibility before the condition.

The possibility after the condition is calculated by extending the membership function to the max value in the direction of the signal direction and then evaluating this modified membership function as in Figure 3.12. This can be described by the equation:

$$P_{after}(a) = \sup_{x \leq a}(\mu(x)) \tag{3.8}$$

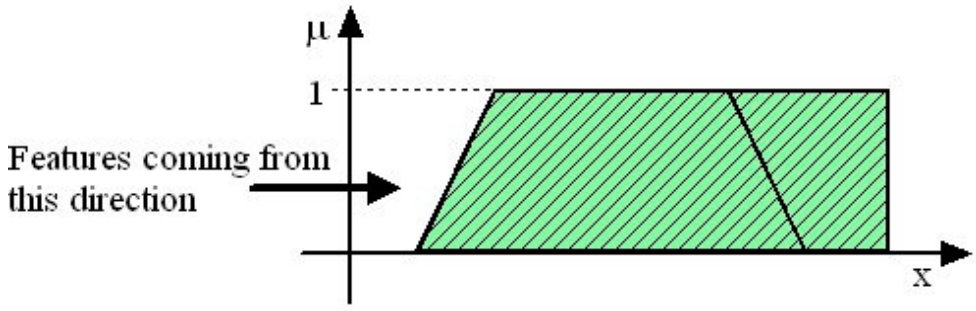

Figure 3.12. Possibility after the condition.

The marking of the input place is the possibility before the condition. The output place marking is the minimum between the possibility before the condition and the possibility after the condition. If the marking at the output place passes the possibility threshold, then the corresponding transition fires. If there are several membership functions describing conditions that must be met for a transition to fire, then the minimum of the output place marking is taken and the threshold is tested.

3.2.2 Fuzzy Classifier as a Mode Identifier

In this section, a classification technique based upon the system dynamics is presented. It is assumed here that every mode has unique features characterizing the dynamics. Since events are sometimes difficult to detect, this module acts as verification to the fuzzy Petri net classifier of the previous section. Mode identification through dynamics will proceed first by extracting features from incoming sensor and control signal data. The incoming feature data is then classified through a fuzzy logic rulebase to determine the mode.

Features are extracted by a feature extraction module and placed into a database. The fuzzy logic classifier loads the required features from the database and then determines the current operating mode through a Mamdani inference engine as shown in Figure 3.13.

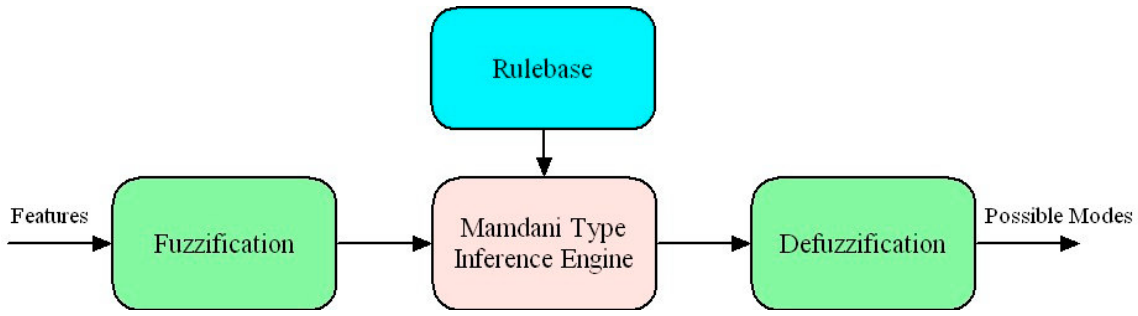

Figure 3.13. The Fuzzy Logic Classifier.

The fuzzification process fuzzifies the incoming features to values in the range of [0,1] through a membership function. This determines the degree of membership that an element belongs to a particular fuzzy set. For example, the fuzzy set "Very High" may have a membership function as shown in Figure 3.14.

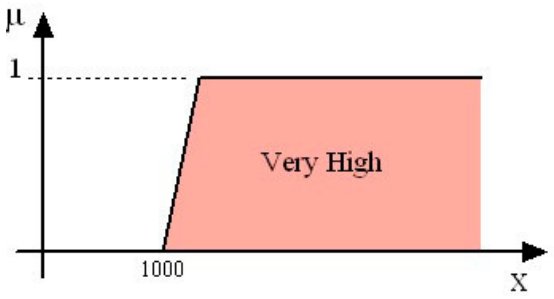

Figure 3.14. Membership function for fuzzy set "Very High."

In general, membership functions can come in a wide variety of shapes, however, the membership function selection has been limited to a generic set of five shapes: triangular, trapezoidal, Gaussian cure, difference of two sigmoids, and bell shaped. Parameters are provided to allow for modification of the position, width, etc. of these shapes for different fuzzy sets.

Expert information about the characteristics of a mode is used to create both the membership functions and the rules to determine the mode. Example of such rules are:

If the *Tank Level Noise Level is High* and the *Tank Level Slope is Large Negative*

then the *Current Operating Mode is Mixer-and-Pump On*

If the *Tank Level Slope is Large Negative*

then the *Current Operating Mode is Pump On*

For each rule, Mamdani type implication is performed which takes the form of Equation 3.9 for the generic rule, If *A* then *B*.

$$\mu_R(x, y) = \max\{\min[\mu_A(x), \mu_B(y)]\} \tag{3.9}$$

where, $\mu_R(x, y)$ is the result of the implication, $\mu_A(x)$ is the input membership function for fuzzy set, *A*, and $\mu_B(y)$ is the output membership function for fuzzy set, *B*.

Output membership functions such as (*Current Operating Mode is Mixer-and-Pump On*) and (*Current Operating Mode is Pump On*) are defined on the universe of discourse within the range [0,100]. This is to convey belief that a rule will correctly determine the operating mode.

After the Mamdani implication has been performed for each of the rules for a particular operating mode, a union (Maximum type, OR) operation is performed upon all the resulting membership functions. This results in a single fuzzy set, $\mu_C(z)$. Following these operations, a centroid type defuzzification is performed upon the resulting membership functions as in Equation 3.10.

$$Z = \frac{\int \mu_C(z) z dz}{\int \mu_C(z) dz} \tag{3.10}$$

3.2.3 Evidence Combiner

The evidence combiner module combines the possibilities of both the fuzzy classifier and fuzzy Petri net to determine the mode. The combination involves taking the maximum of the possibilities of each mode from the separate modules. The mode corresponding to the maximum value from this result then is chosen to determine the mode.

3.2.4. The Mode Identification Software Algorithm

The mode identification software algorithm is shown in Figure 3.15. The past operating mode is read in from a stored location to initialize mode of the system. The fuzzy classifier module is then run to determine the new operating mode. If this is the first run of the mode identification module then, only the fuzzy classifier is run to determine the mode, else the fuzzy Petri net proceeds to identify the mode based on events. Next, the evidence is combined from both the fuzzy classifier and fuzzy Petri net and the resulting operating mode is stored in the database.

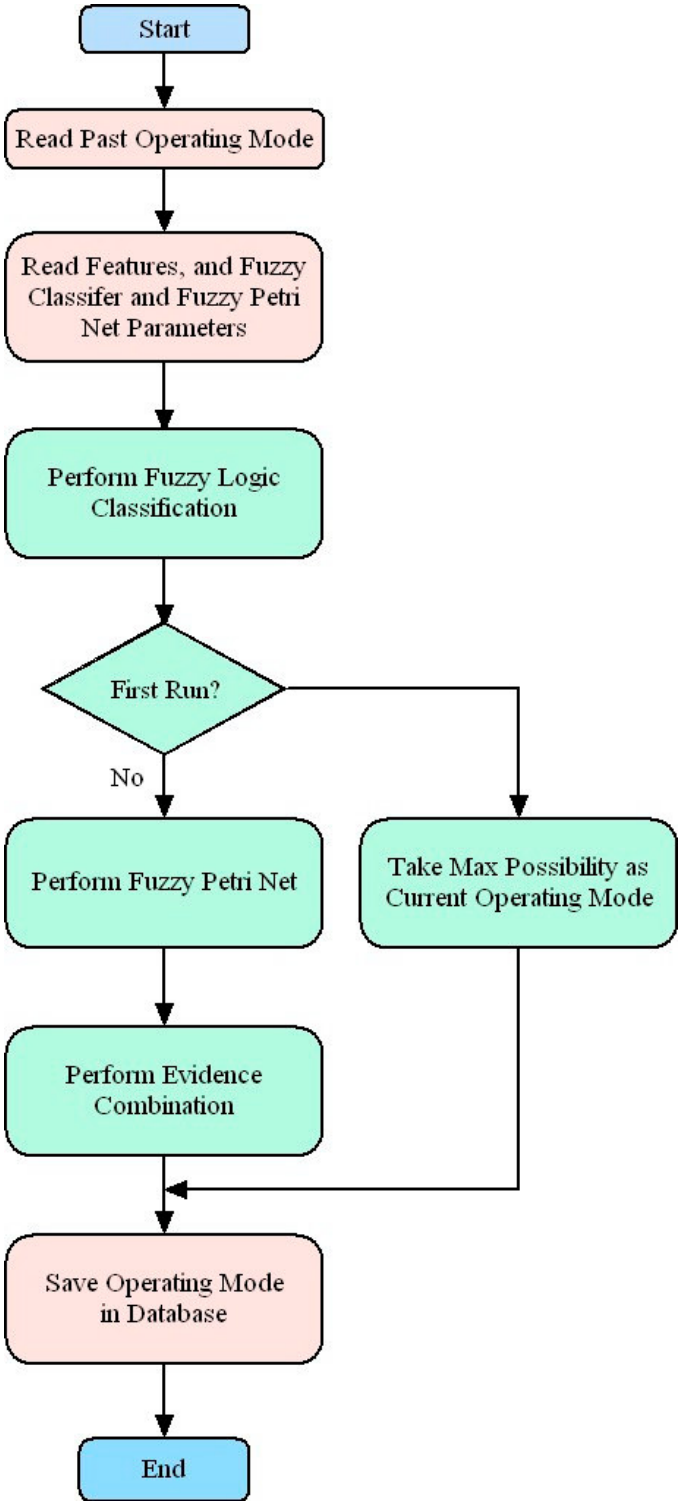

Figure 3.15. The mode identification operational chart.

3.3 System Level Reconfiguration

In this section, we examine how to further reconfigure the control system in the presence of system failures. Such control schemes are called fault tolerant control systems and are found in many applications such as aircraft, naval vessels space vehicles, and, structures. The overall goal of a reconfigurable control scheme is to maintain system stability and retain acceptable performance in the presence of failures. Therefore, for manufacturing systems, we wish to design a control architecture that allows the system to continue operation in the presence of failures while maintaining required performance levels in a degraded mode of operation. .

Consider our model dynamics, as shown in Figure 3.16. At the top level, the supervisory control determines the paths the products take through the system, and at the lower level, the hybrid dynamics between the subsystem operations and the time-driven dynamics are displayed. Thus, the control authority can be distributed through the product paths, the operations, and the low-level controllers. In addition, the object-oriented model links a subsystem's submodels, and thus, we can distribute the available control authority between the submodels. In the following sections, we provide solutions to the following problems:

1) How can we reconfigure the product paths through a manufacturing system in the presence of a subsystem failure?
2) How can we reconfigure the operation paths for a particular subsystem when certain operations are no longer viable?
3) How can we reconfigure the event and state variable linkages between object dynamic submodels?

4) How can we reconfigure the low level control authority between the dynamic submodels due to small disturbances such as changes in the environment or unexpected model inaccuracies?

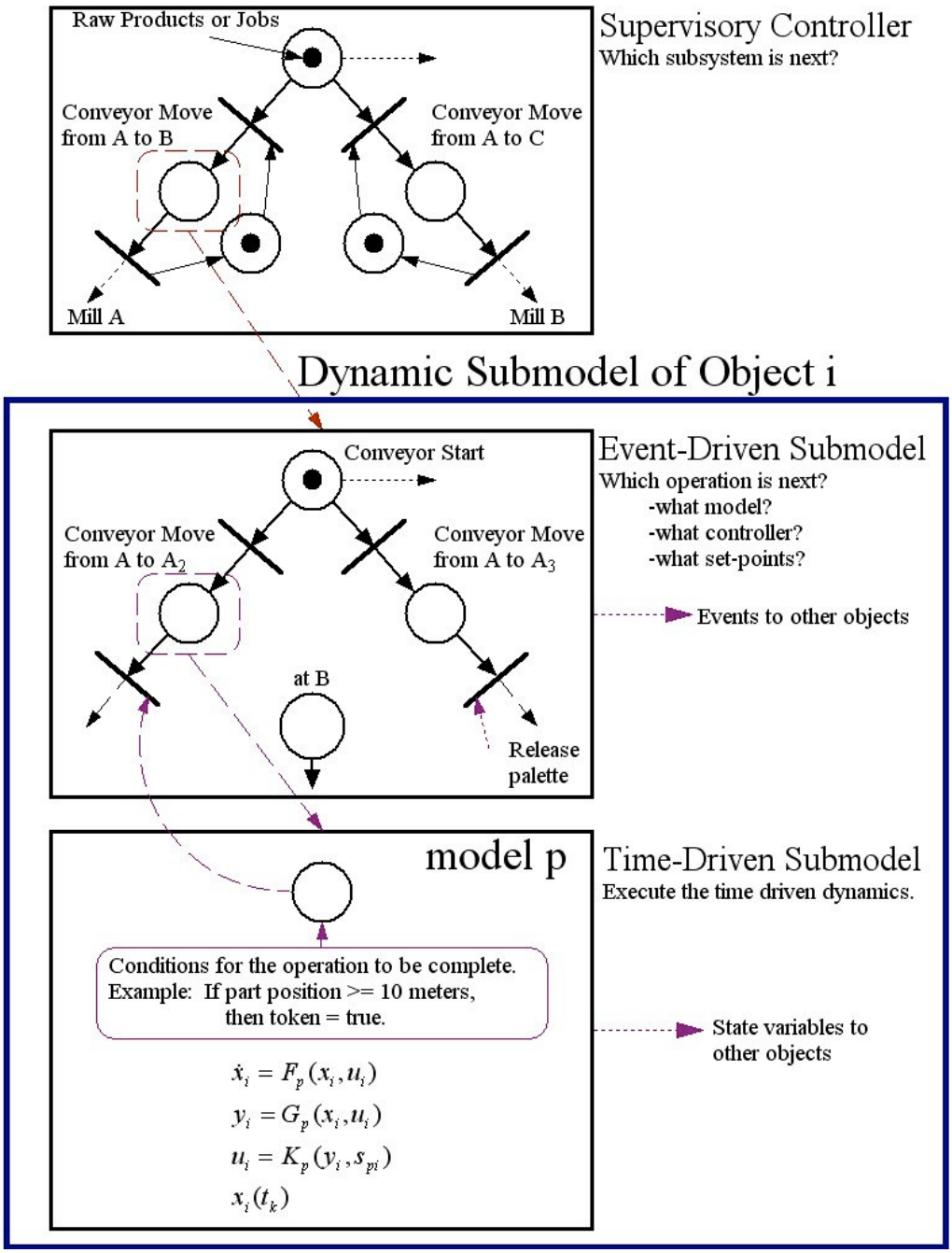

Figure 3.16. The model structure.

3.3.1 Product Path Reconfiguration

The rerouting of products through different paths is performed when certain subsystems are no longer functioning. For example, if a mill motor is not operational, then that mill may no longer be able to be used to perform tasks upon the product, and another mill, if available, should be found and used instead. The supervisory Petri net, described in Section 3.1.5, is used to perform this rerouting by identifying the most suitable route for the product.

Since each token in the supervisory Petri net can be associated with a product as it travels through the manufacturing system, routes can be chosen by selection of transition sequences in the Petri net. In fact, the sequence of transitions can be reduced to just those Petri net places where choices can be made.

Figure 3.17 shows such a scenario where a token can choose to move to different places. A simple example of this is a manufacturing system with parallel conveyor belts

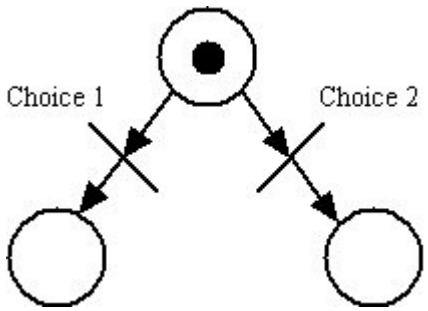

Figure 3.17. Choices in a Petri Net.

each performing the same process on the product. In more complex cases, the supervisory Petri net can include emergency paths where parts move back and forth through the manufacturing system to find replacement subsystems for the faulty ones.

Therefore, for each Petri net token (or product) we associate a route, that is composed of the transition choices:

$$T_{path} = (c_1, c_2, ..., c_N) \tag{3.11}$$

To find the optimal route, we select the path that achieves the desired optimality conditions such as minimizing time, minimizing/maximizing set-points levels, etc. Let us assume that the system operating set-points are set to perform the tasks as quickly as possible without degrading product quality. Therefore, each possible path can be associated with a total time for product completion. When a faulty subsystem is detected, supervisory Petri net transitions leading to this subsystem are disabled. Then all feasible paths of each token/product are found, and the best path for each token is selected by choosing the path that meets the desired optimality constraints.

To determine the possible paths, consider the simple Petri net supervisor in Figure 3.18.

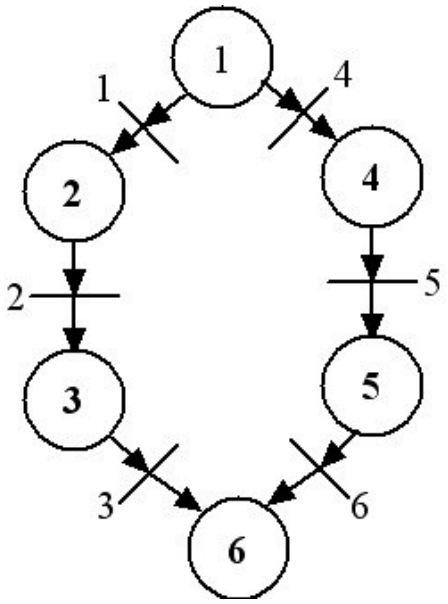

Figure 3.18. Simple Petri net supervisor.

The goal is to move raw product materials from place 1 to place 6. Let us say places 3 and 5 represent the same milling operations. Reachability analysis can tell one whether or not it is possible to move the raw product materials from place 1 to place 6, but it is clear from this simple example, that it is possible if either transition 2 or 5 are disabled as long as the token starts in place 1. To determine the feasible product paths it is necessary to examine the input and output matrices of the Petri net.

$$M_{Input} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.12}$$

$$M_{Output} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

The choices can be found in the input matrix by examining rows that have more than one, 1 element. The input matrix in the example shows that the first row has a choice between place 2 and place 4 upon the next iteration of the Petri net. One can simulate the Petri net through the equation below to determine the possible paths:

$$\mu(k+1) = \mu(k) + (M_{Output} - M_{Input})T_k, \tag{3.14}$$

$$\mu(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.15}$$

where $\mu \in Z^6$ is a column matrix representing the current state of the Petri net and $T_k \in Z^6$ is a column matrix representing the transition that is fired at iteration, *k*. The sequence of Petri net states are then stored for each path for optimal selection.

If a failure occurs in the mill at place 3, then the extent of the failure and the number of products will determine the path of the object. If mill 3 completely fails, then there is only one path that can be chosen. If mill 3 can be reconfigured so that operation on that path is still viable and the number of products is greater than 1, then it may be desirable to continue to utilize this path. A simple optimization function could be:

$$\min J(p) = An_p + BT_p + C\sum_{k=1}^{N_p} \left\| s_{pk} \right\| \tag{3.16}$$

where p is the path index, n_p is the number of steps through the Petri net for a path, T_p is the total path time, N_p is the number of set-points, and s_{pk} are the set-points of subsystem k and *A*, *B*, and *C* are constants chosen by the designer. Such an optimization function describes a trade off between set-points and time. Thus, the steps for the path reconfiguration are as follows:

1) Determine the viable paths of the Petri net. Failures may reduce these paths.
2) Find the optimal route for the raw materials by solving an optimization problem based on these paths.

3.3.2 Functionality Submodel Optimization

The functionality submodel included in the design of the Object-based World Model determines the order or flow of operations. The Object-based Hybrid Dynamic Model, as described in Section 3.1.5, in fact, includes the functionality submodel within the

dynamic submodel's Petri net. Each Petri net place in the dynamic submodel is associated with an operation and the operation paths can be chosen in the same manner as in Section 3.3.1. However, the reconfiguration is slightly more complex.

There are times when a failure prevents some future operations to be performed by the subsystem, and these operations must be avoided to prevent further damage to a machine or product. For example, if a mill spindle motor fails, then the mill cannot perform the required operations upon a product. In this case, we wish to skip over operations (see Figure 3.19) requiring the cutting of product material and go to directly to the operation that removes the product from the current mill to find another mill that can perform the desired operations. This type of problem can be handled by simply reinitializing the token placement within the dynamic submodel's Petri net once the failure has been detected and preventing transitions leading to the non-functioning mill from being fired.

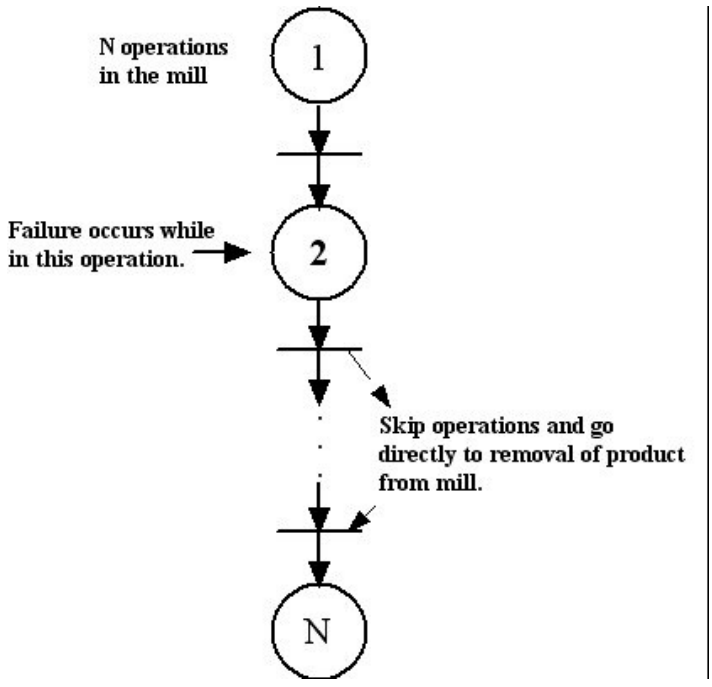

Figure 3.19. Skipping operations due to a severe failure.

There are other cases where one can recover from a failure by adding operations. For example, consider a mill bit breaking. In this case, it may be possible for the mill to retrieve a new bit from storage. Thus, once the failure has been detected, the current operation must be halted and the current Petri net state stored. A separate Petri net then takes over operations to retrieve a bit. Once these operations have been performed, the Petri net returns to the previous unfinished operation if appropriate. Figure 3.20 shows these recovery operations. In the case of the mill, it may be necessary to start from an earlier state than the one stored in memory. This requires some faults to be associated with a fault recovery Petri net and a Petri net re-initialization routine.

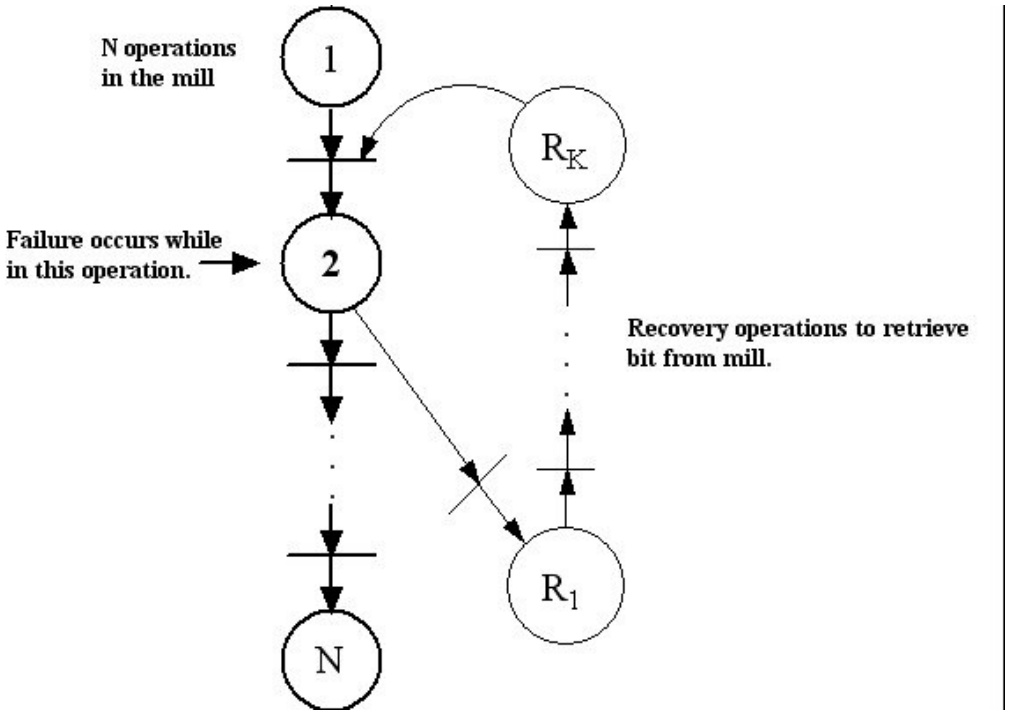

Figure 3.20. Additional operations to recover from a failure.

The steps to achieve reconfiguration are therefore designed into the Petri net description of the hybrid model before operation and must be taken into account in the selection of the optimal path if considering worst case scenarios.

3.3.3 Object-based Hybrid Model Reconfiguration

The reconfiguration ideas presented above can be utilized within the OHM dynamic submodel for each object fairly easily by appropriately masking events that are not to be fired and providing alternate controllers to handle failure if the current controller is not sufficiently robust. A single subsystem itself can be divided into several objects. As was shown in Section 3.1.5 (Figure 3.7), each of these objects can have separate Petri nets and continuous time-driven equations modeling part of the subsystem. The time-driven equations used in simulation are governed by the associated Petri net state, and the flow of operations are governed by the event mask, failure type, state variables, and Petri net states of the supervisory and subsystem objects.

Consider a robot with 4 degrees of freedom where there is a robot base object, arm 1 object, arm 2 object, and end effector object as in Figure 3.21. If there is a joint stuck failure, then by designing an event mask we can force the failed joint to go to the brake state and remain there until the joint has been repaired.

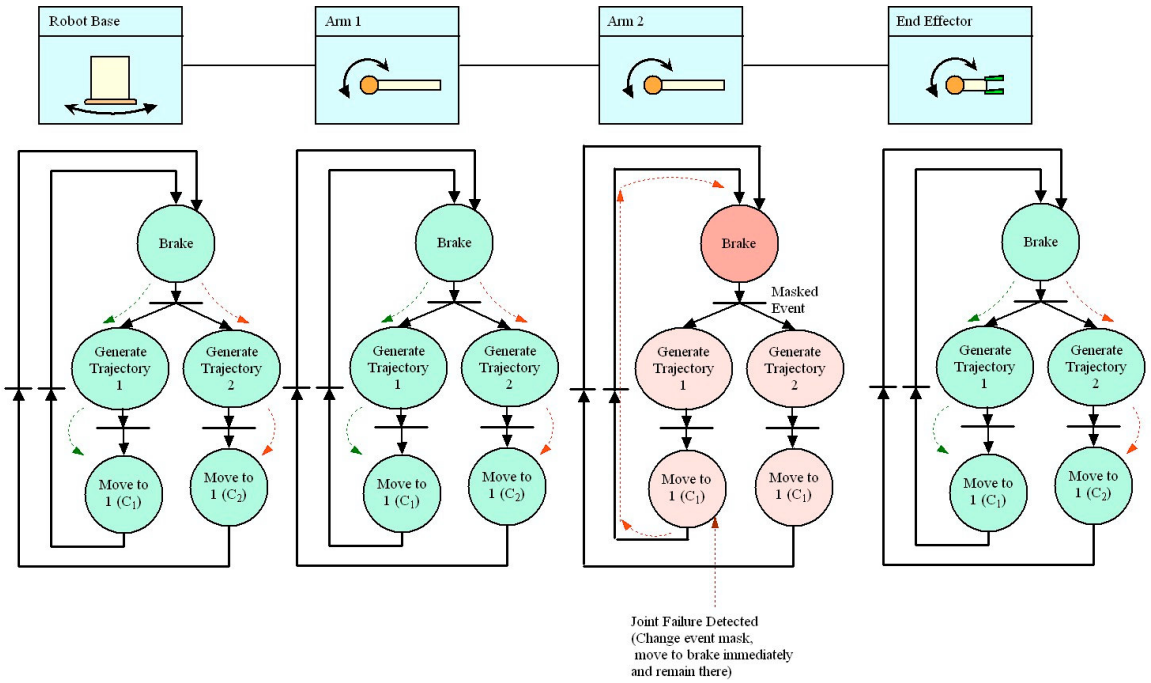

Figure 3.21. Robot joint failure example.

This type of robot failure causes the trajectory to be constrained so new trajectories should be created for joints that require it. In some failure cases, the low-level controller must be modified as well as Figure 3.21 shows with C_1 and C_2 controllers.

3.3.4 Adaptive Control of the Low Level Controllers

Instability can occur when the parameters describing a system change. Parameter changes can occur for a number of reasons. For example, a robot picking up different types of parts changes the dynamics of the robot, or the weight on a conveyor belt can vary, etc. An adaptive control algorithm attempts redesign the controller using a direct Model Reference Adaptive Control (MRAC) scheme to achieve proper tracking.

We now wish to show a possible adaptive control law for tracking a trajectory. Let us assume that the time-driven dynamics of the system can be represented in linear form as:

$$\dot{x}_i = F_{\mu_i}(x_i, u_i) = A_{\mu_i} x_i + B_{\mu_i} u_i \tag{3.17}$$

$$y_i = G_{\mu_i}(x_i, u_i) = C_{\mu_i} x_i + D_{\mu_i} u_i \tag{3.18}$$

$$u_i = K_{\mu_i}(y_i, s_{\mu_i}) \tag{3.19}$$

$$x_{i\,\mu_i}(t_k) \tag{3.20}$$

$$\mu_i[k+1] = \mu_i[k] + (D_i^+ - D_i^-) E_i T_k \tag{3.21}$$

where for subsystem i, $x_i \in R^n$ is the continuous state vector, u_i is the input vector, $y_i \in R^m$ is the output vector, $A_{\mu_i} \in R^{n\times n}$, $B_{\mu_i} \in R^{n\times r}$, $C_{\mu_i} \in R^{m\times n}$, $D_{\mu_i} \in R^{m\times r}$, K_{μ_i} is the controller mapping, s_{μ_i} are the set-points, $x_{i\mu_i}$ is the initial state, μ_i is the Petri net state, D_i^+ and D_i^- are the output and input matrices, respectively, E_i is the event mask matrix, and T_k is the transition firing vector.

A known adaptive tracking law now can be used [32]. If we consider a direct model reference adaptive control scheme with reference model,

$$\dot{x}_i^M = A_{\mu_i}^M x_i^M + B_{\mu_i}^M r_{\mu_i} \tag{3.22}$$

where, r_{μ_i} is the bounded reference input vector, and x_i^M is the desired trajectory, we obtain the following control laws:

$$u_i = -K_{\mu_i}(t) x_i + L_{\mu_i}(t) r_{\mu_i} \tag{3.23}$$

$$\dot{K}_{\mu_i} = B_{\mu_i}^{M\,T} P_{\mu_i} e_{\mu_i} x_i^{\,T} \operatorname{sgn}(l_{\mu_i}) \tag{3.24}$$

$$\dot{L}_{\mu_i} = -B_{\mu}^{M\,T} P_{\mu_i} e_{\mu_i} r_{\mu_i}^{\,T} \operatorname{sgn}(l_{\mu_i}) \tag{3.25}$$

where,

$$P_{\mu_i} A_{\mu_i}^M + A_{\mu_i}^{M\,T} P_{\mu_i} = -Q_{\mu_i}, Q_{\mu_i} > 0 \tag{3.26}$$

$$\operatorname{sgn}(l_{\mu_i}) = \begin{cases} 1 & L^*_{\mu_i} > 0 \text{ given } B_{\mu_i} L^*_{\mu_i} = B^M_{\mu_i} \\ -1 & L^*_{\mu_i} < 0 \text{ given } B_{\mu_i} L^*_{\mu_i} = B^M_{\mu_i} \end{cases} \tag{3.27}$$

$$e_{\mu_i} = x_i - x_i^M \tag{3.28}$$

One restriction on this method is that the structure of B_{μ_i} must be known. Also there is no guarantee that $L^*_{\mu_i}$ is positive or negative definite.

3.4 Product Level Reconfiguration

In some cases, control reconfiguration techniques adjusts set-points of controls to increase the performance of a system due to system deviations. In this thesis, we wish to detect variations in manufactured products outside nominal statistical measures and utilize a control reconfiguration scheme to improve the product quality.

Common automated techniques used to verify product quality in manufacturing processes utilize feature data from sensors such as 2-dimensional or 3-dimensional cameras, weighing balances, etc. to verify the shape, structure, color, etc. of finished products. For example, a camera takes a image of a plastic part and an algorithm extracts color features from this image in order to verify there are no burn marks upon the surface of the part. The comparison could be as simple as detecting if a certain color or colors are present. By extracting statistical information about the features more information about the manufacturing process and the product is generated and a reconfigurable control scheme is possible which can improve product quality automatically. For example, consider a feature describing the depth of a cut by a CNC mill that is supposed to have a depth of 5 cm. If depth feature data is gathered from numerous acceptable product samples, we can construct a template histogram displaying the frequency of

occurrence of depth for the sample set as shown in Figure 3.22. One can then compare this template histogram with a histogram generated from an operating manufacturing process by comparing appropriate statistical measures.

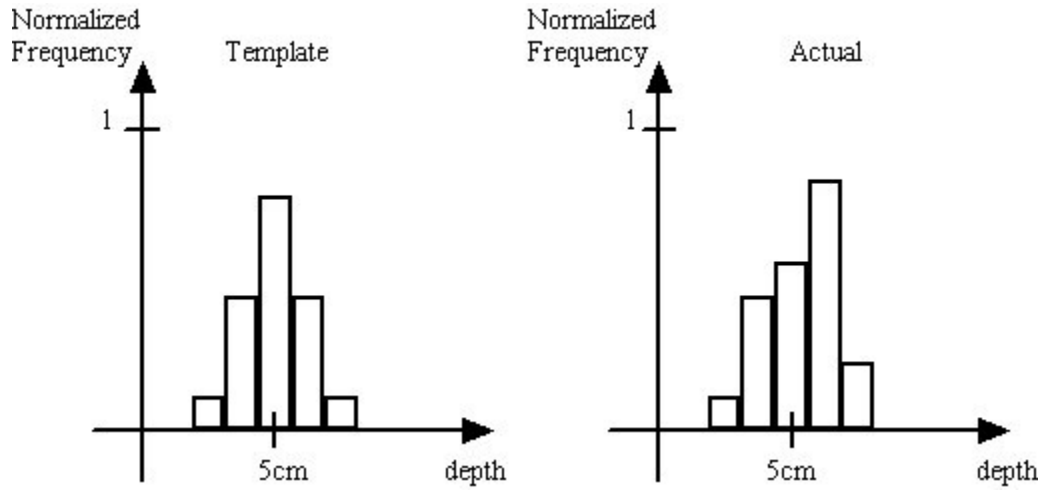

Figure 3.22. Histograms of template and actual depths of cut.

Figure 3.23 shows the architecture of the reconfigurable control scheme explained in this section. The methodology presented in this section consists of a number of steps towards the goal of the reconfigurable control:

1. Gathering and processing of sensor data into features.
2. Creating template histograms of feature data representing good products.
3. Comparing template histograms with generated histograms during operation.
4. Generating feedback signals to adjust set-points to improve product quality when the histogram comparison is outside normal parameters.

In order to perform these steps efficiently, a number of intermediate steps must be performed to optimize the performance of the reconfiguration process.

1. Features must be properly selected from a "bank" of features. This is the operation known as feature selection.
2. Proper comparison must be performed between template and generated feature histograms in order to provide correct error feedback information to the set-points.
3. The detection of unacceptable products must be performed first so that unnecessary computation is not performed.
4. The control reconfiguration mechanism must be optimized to provide efficient and correct feedback to the set-points.

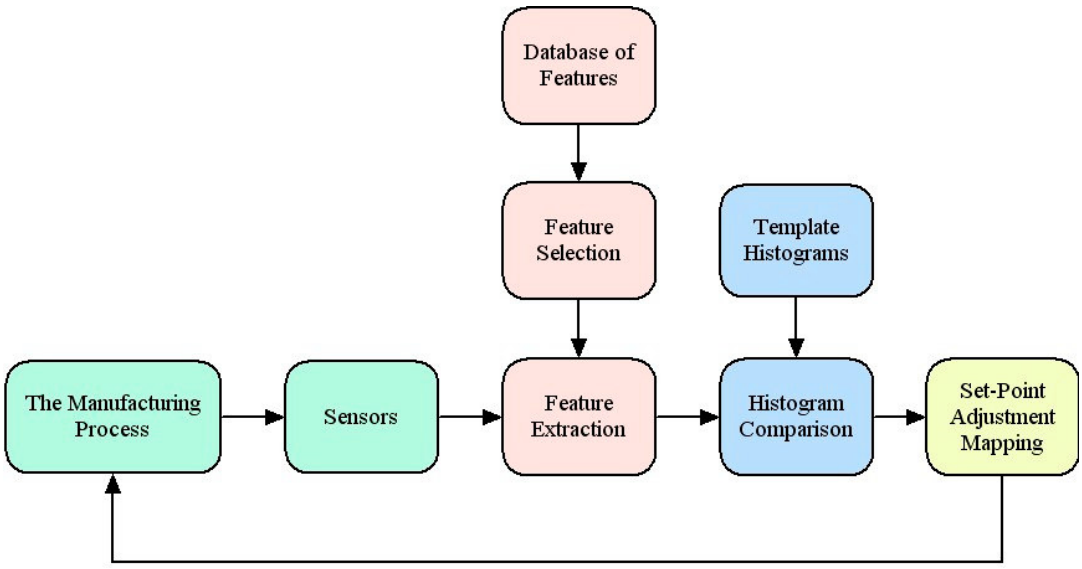

Figure 3.23. The product control reconfiguration architecture.

3.4.1 Offline Selection of Features

The feature selection method used in this thesis is an offline process of determining a set of "best" features from a large set of generic features. A standard genetic algorithm with

a certain fitness evaluation function is used to select the best features and is described next [19].

Chromosomes are constructed with each bit corresponding to a separate feature. For example, 01010, would represent features F_2 and F_4 as being selected. A number of chromosomes are randomly created for the initial population C_1, C_2, …, C_p, of p chromosomes. Let C_k^l denote the lth bit of the kth chromosome and l = 1,2, …, m, where m is the number of features. For example, if C_2 = 01010 where $C_2^1 = 0$ and $C_2^4 = 1$. From this initial population, mutation and crossover operations are used to create n, child chromosomes, C_{p+1},…, C_{n+p}.. A fitness metric is then used to select the best p chromosomes from the combined initial and child populations. A fitness metric is used that incorporates the accuracy of the fault classification and usefulness in the reconfiguration operation to determine the fitness of each chromosome. This metric, G_k, is defined as:

$$G_k = F_{within}^k + F_{between}^k + \frac{\Psi}{N_k}, k = 1,...,(n+p) \tag{3.29}$$

where,

$$F_{within}^k = 1 - A\sum_{i=1}^{m} C_k^i \sum_j \left(F_i^j - F_i^C\right)^2 \tag{3.30}$$

$$F_{between}^k = B\sum_{i=1}^{m} C_k^i \sum_{j=1}^{m} \sum_{q,j\neq i} \left(F_j^q - F_i^C\right)^2 \tag{3.31}$$

$$N_k = \sum_{i=1}^{m} C_k^i \tag{3.32}$$

$$F_i^C = \frac{\sum_j F_i^j}{\phi_i} \tag{3.33}$$

where, A and B are real valued constants, C_k^i is the i^{th} bit of chromosome k, F_i^j is the j^{th} feature data of the feature type i, N_k is the number of features in the chromosome, F_i^C is the cluster center of feature type i, and ϕ_i is the total number of feature data for feature type i. The data is divided into cluster centers, F_i^C, associated with the particular product quality defect. F_{within}^k measures the spread of feature data associated with a cluster center. $F_{between}^k$ measures the distance from the a cluster center from data associated with different clusters. $\frac{\Psi}{N_k}$ represents the need to find the fewest number of features. It is desirable to maximize G_k to select the best features.

A roulette wheel technique is used to select the best features by first determining the fitness probability for each chromosome through the equation:

$$E_k = \frac{G_k}{\sum_{j=1}^{n} G_j}, k = 1,2,...,(n+p) \tag{3.34}$$

Next, the cumulative fitness probability is then calculated as such:

$$H_k = \sum_{j=1}^{k} E_j \tag{3.35}$$

The roulette wheel is spun by generating a random number, R, between [0,1] and selecting the appropriate chromosome. If $R < H_1$ then chromosome 1 is selected. If $H_{k-1} \leq R < H_k$ for $k \geq 2$, then chromosome k is selected. The roulette wheel is spun p times to yield a new population and the process begins again with this new population as the initial population. The genetic algorithm is terminated when a goal fitness value has been reached.

3.4.2 Histogram Construction and Template Histograms

As was mentioned previously, we wish to construct histograms of feature data in order to determine the product quality of manufactured items. To create these histograms, feature data is necessary for the same product type but on different runs of the manufacturing process. Therefore, we have a set of feature data, F_k^j, where *j=1,2,…,n* represents the product number that was evaluated and *k=1,2,….,m* represents the feature type. For each feature, we wish to construct a function, H_k, which maps feature values to a normalized frequency of occurrence (i.e. a normalized histogram).

$$F_k^N = H_k(F_k^j) \tag{3.36}$$

To construct the normalized histogram, the feature axis is divided into half-closed, half-open intervals called bins that are nonintersecting but covering a continuous portion of the feature axis. For example, the depth feature axis as shown in Figure 3.21 might be divided into bins as [0 cm, 2 cm), [2 cm, 4 cm), [4 cm, 6 cm), [6 cm, 8 cm), and [8 cm, 10 cm). Let the intervals of these bins be defined as u_k^i where $i = 0,1,...,b$ and $k = 0,1,...,m$ is the feature type, so that the intervals are defined as $[u_1^0, u_1^1), [u_1^1, u_1^2), ..., [u_1^{b-1}, u_1^b)$ for a total of b bins. Bins do not necessarily have to be of the same width, but in our case we will assume this is so (i.e. $\left\| u_k^{i-1} - u_k^i \right\|$ is constant). An integer count value, I_k^c, initialized to 0 is associated with bin, $[u_k^c, u_k^{c+1})$, where *c = 0,1,…,b-1*. If F_k^j falls within one of these intervals, then 1 is added to I_k^c the appropriate bin. Since these sets are nonintersecting, there is no conflict in which bin count value to perform the addition. After all feature data of F_k^j has been used to calculate the $I_k^c{}'s$ in this manner, then all I_k^c count values are divided by total number of data to normalize.

The domain of H_k then is the union of the bin intervals and the range is the normalized count values.

Selecting the number of bins is an issue that must be taken into account. The number of bins determines how well the statistics will be in reflecting the ideal, continuous distribution. Determining how many histogram bins should be used for estimating distributions is a problem in non-parametric statistics. It has been shown [53] that the optimal histogram bin size, which provides the most efficient, unbiased estimation of the probability density function, is achieved when :

$$bin\ width = 3.49\sigma N^{-\frac{1}{3}} \tag{3.37}$$

where σ is the standard deviation of the data and N is the number of data available. In practice, a estimated standard deviation is used to determine the bin width. Therefore, once the sample size is set, the bin width, and thus the number of bins is set through this equation.

Template histograms representing good products are created by recording features of runs of good products and then constructing a histogram for each feature.

3.4.3 Histogram Comparison Features

To determine the product quality, template information of good parts are compared with incoming data of new parts and an error vector is created. The methodology is as follows. First histograms for the features extracted from good parts are stored. These are called the template feature histograms and will be compared with histograms derived from incoming data of the operating process.

Comparing distributions of data is an important goal for several applications. For example, it may be required to determine whether the frequency distribution of a test

sample is significantly different from a control sample. Statistical measures are commonly used to detect changes in a system and provide feedback information for corrective action in stabilizing instruments. The statistical measures used in histogram comparison are listed as follows:

1. *Estimated Mean*

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N} \tag{3.38}$$

The mean is the most probable event within the distribution. For some distributions, the mean may not convey much information. The estimated means of two distributions can be simply compared through subtraction of the estimated means.

2. *Estimated Variance*

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N} \tag{3.39}$$

The variance measures the confidence in the mean and the spread of the distribution. If a target variance is specified and the running variance is smaller, then the products should represent good products. If the variance is larger than the target variance, then some adjustments must be made. The estimated variances can be compared by subtracting the variances of two distributions.

3. *Estimated Skewness*

$$\gamma = \frac{\sum_{i=1}^{N} (x_i - \mu)^3}{N\sigma^3} \tag{3.40}$$

Skewness vanishes for symmetric distributions and is positive (negative) if the distribution develops a longer tail to the right (left) of the mean $E(x)$. It measures the

amount of spread of the distribution in either direction from the mean. Subtracting the skewness of two distributions compares the estimated skew measurements.

4. *Estimated Kurtosis*

$$kurtosis = \frac{\sum_{i=1}^{N}(x_i - \mu)^4}{N\sigma^4} \tag{3.41}$$

Kurtosis measures the contribution of the tails of the distribution. It is possible for a distribution to have the same mean, variance, and skew, and not have the same kurtosis measurement. The estimated kurtosis can be compared by subtracting the kurtosis of two distributions.

5. *Taguchi Variance*

$$S_T = \sigma^2 + (\mu - T)^2 \tag{3.42}$$

The Taguchi variance is a measurement that separates the differences in target and observed mean and the variance of the observed samples. This variance allows one to determine whether the problem is due to a shift in mean or due to an increase in variance. To compare two distributions, the Taguchi variance is measured for each distribution and then subtracted.

6. *Total Deviation*

$$TD = \sum_{i=1}^{B} \max\left|x_i - y_i\right| \tag{3.43}$$

This is a measure of the area of the difference between the distributions.

These statistical measures represent how well the template and online generated histograms compare and will be used to adjust set-points to bring the quality back within acceptable parameters. Since this computation will only take place when a problem with

product quality has been detected, there is only additional computational burden when necessary.

3.4.4 ANFIS Reconfiguration

The Adaptive Network-based Fuzzy Inference System (ANFIS) structure is used to map the input histogram comparison features to the output set-point adjustments. The ANFIS structure is basically an adaptive network that is functionally equivalent to a fuzzy inference system. The reasons behind using this structure are ANFIS permits learning of the mapping through training and allows some interpretation of the behavior through the rules which are created.

The ANFIS architecture network structure can be understood by considering two fuzzy if-then rules of Takagi-Sugeno type:

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1 x + q_1 y + r_1$

Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2 x + q_2 y + r_2$

Where, x and y are inputs, A_i and B_i are input membership functions, f_i are the fuzzy outputs, and p_i, q_i, and r_i are parameters used to calculate the output.

The defuzzified output can be written as:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \overline{w}_1 f_1 + \overline{w}_2 f_2 \tag{3.44}$$

The ANFIS structure is shown in Figure 3.24. We now describe the layers of the ANFIS structure.

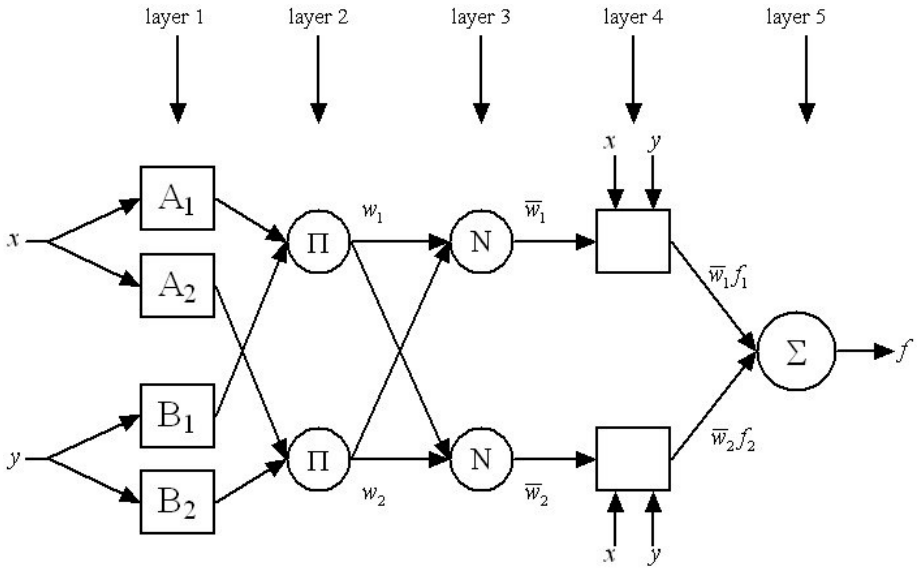

Figure 3.24. The layers of the ANFIS structure.

Layer 1: Every node i has a fuzzy membership function associated with it:

$$L_i^1 = \mu_{A_i}(x) \tag{3.45}$$

$\mu_{A_i}(x)$ is a membership function which specifies the degree which x satisfies A_i and outputs real numbers between 0 and 1, inclusive. A_i and x can be replaced with the appropriate letter associations with the input.

Layer 2: Every node in this layer multiplies the outputs of Layer 1 as:

$$L_i^2 = w_i = \mu_{A_i}(x) \times \mu_{b_i}(y), i = 1, 2. \tag{3.46}$$

This output is the firing strength of the rule.

Layer 3: Every node in this layer computes the ration of the firing strengths as:

$$L_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2}, i = 1, 2. \tag{3.47}$$

Layer 4: Every node in this layer calculates the consequent parameters as:

$$L_i^4 = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i), \; i = 1, 2. \tag{3.48}$$

Layer 5: This final node combines the results for layer 4 to give the defuzzified output:

$$L_i^5 = \sum_i \overline{w}_i f_i \tag{3.49}$$

The training of this adaptive network uses a hybrid learning algorithm consisting of gradient descent and least squared error.

The commonly used software tool called MATLAB (by The MathWorks, Inc.), provides functions *genfis* to train the structure of the network and *ANFIS* to realize the input/output mapping. The ANFIS network is trained on the histogram comparison features and appropriate set-point adjustments for specific faults. When used while the manufacturing system is operational and a product defect is detected, the histogram comparison features are calculated and fed to the ANFIS mapping to produce the outputs, ΔS_i, which are the adjustments necessary to the set-points. The set-points are modified by adding the adjustment factor to the current set-points as:

$$S_i^+ = S_i + \Delta S_i \tag{3.50}$$

where, i is the set-point index, S_i^+ are the modified set-points, and S_i is the current set-point.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 The Experimental Testbed

The testbed manufacturing system (shown in Figure 4.1) is composed of many system components from a wide variety of manufacturers. A central, circular conveyor belt transports palettes and parts from station to station until the required process is complete. The speed and direction of the conveyor belt is controlled by an Opto22 Ethernet I/O PLC. The conveyor belt is surrounded by several sensors (photoelectric, optical, and proximity) used to detect parts and palettes. There are also pneumatic palette stop devices to keep palettes at stations while performing operations. The sensors and palette stop devices feed into a Honeywell Smart Distributed System field bus that is monitored and controlled by a PC computer through use of a Think and Do interface program. A Morgan press, vertical, plastic injection molding machine creates square parts through a mold to be machined by a Light Machines milling station. The Morgan press is controlled manually, but electronic valves could adjust pressures, temperatures and timing of the process if they were available. The Light Machines milling station is controlled by a numerical control software package to "mill out" parts designed through CAD/CAM. A Pulnix camera and Imagenation PXC frame grabber capture 2D color images of the part. A 4DI camera system utilizes three cameras positioned at different angles and a laser to create structured light to capture depth data about the product. A weight scale measures the mass of products. Two Robotech robot arms with 4 degrees-of-freedom each are used to transport parts from and to the conveyor belt and various stations. All the devices are interfaced through a Visual Basic program for simple

scheduling operations with the exception of the Morgan press. 4 PC computers are networked to communicate sensor readings and control settings.

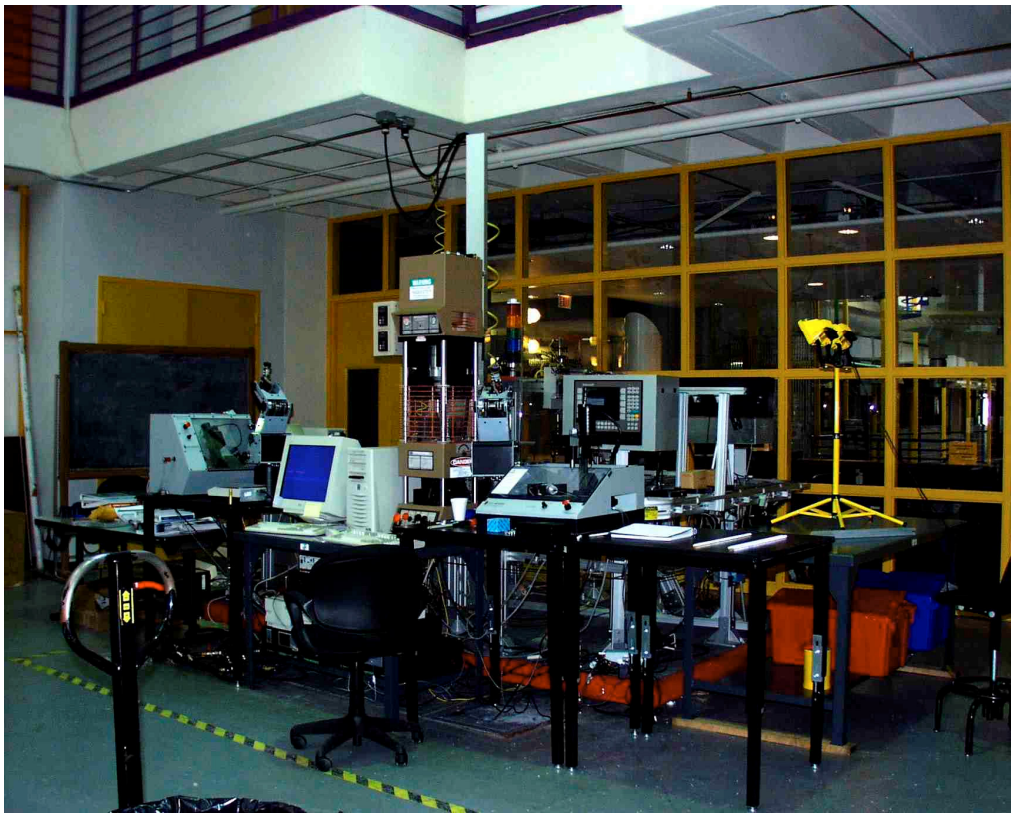

Figure 4.1. The testbed manufacturing system.

4.2 The Modeled Manufacturing System

The control reconfiguration results come from a mixture of simulations performed upon a model and the real system data. A model was incorporated to demonstrate the proposed algorithms, and also provides redundant subsystems that are not included in the real system. The modeled system is similar to that of the testbed system with a circular conveyor belt, a vision system, two mills, two arm robots, and a plastic injection molding machine. Figures 4-2 and 4-3 show the layout of the system and the relative spacing.

The goal of the manufacturing system is represented in the following steps:

1. Take the raw materials from the plastic injection molding machine.
2. Inspect the raw materials for defects.
3. Mill the part.
4. Inspect the milled part for defects.
5. Place the finished product in storage.

There are redundant systems such as the two robots and two mills to demonstrate the rerouting capabilities of the model. Both of these systems are considered to be identical except for the commanded positions of the robots relative to the conveyor belt, plastic injection molding machine, mill, and storage positions. Moreover, product inspection results are only shown for the raw materials and the robots are used to demonstrate the control reconfiguration process.

The first section that follows shows the results from the system level reconfiguration. This chapter concludes with results from the product level reconfiguration.

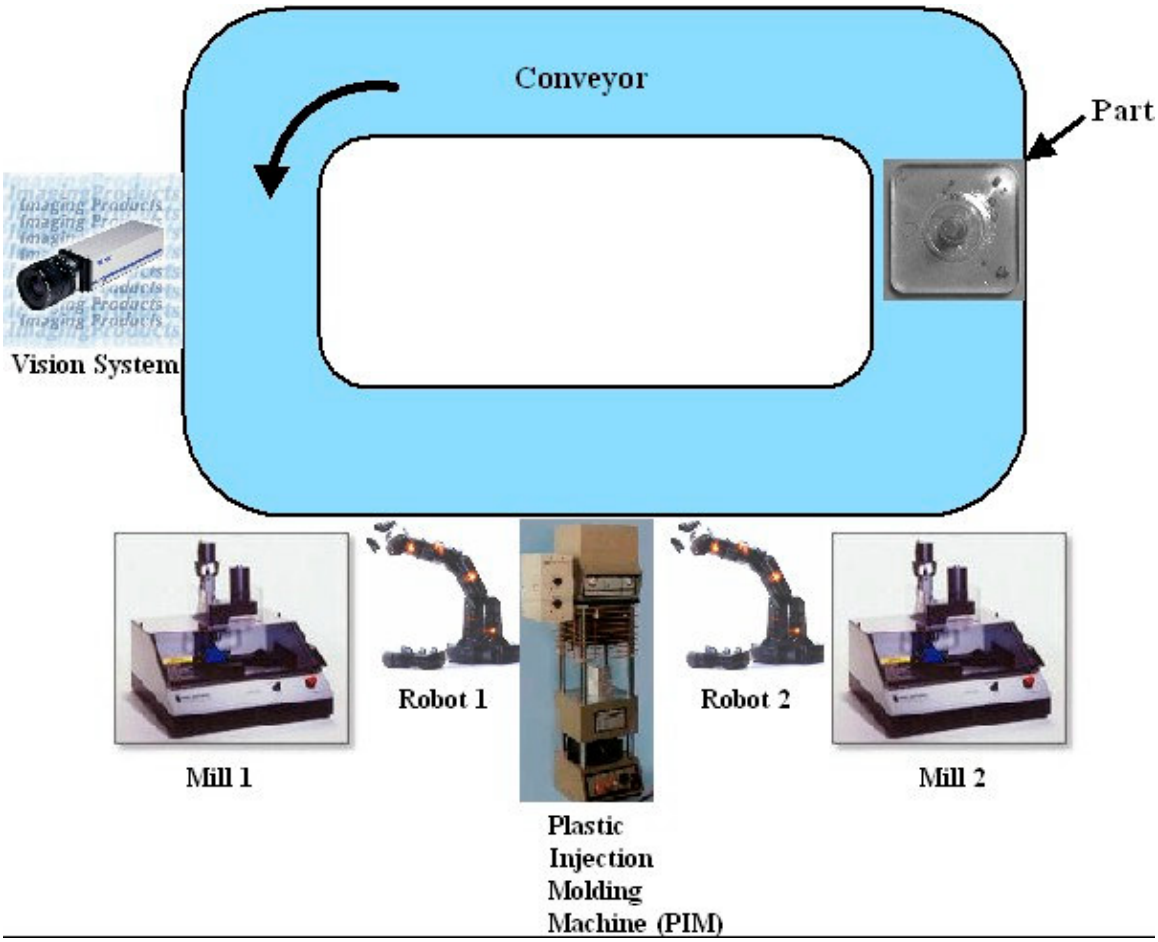

Figure 4.2. The manufacturing system model.

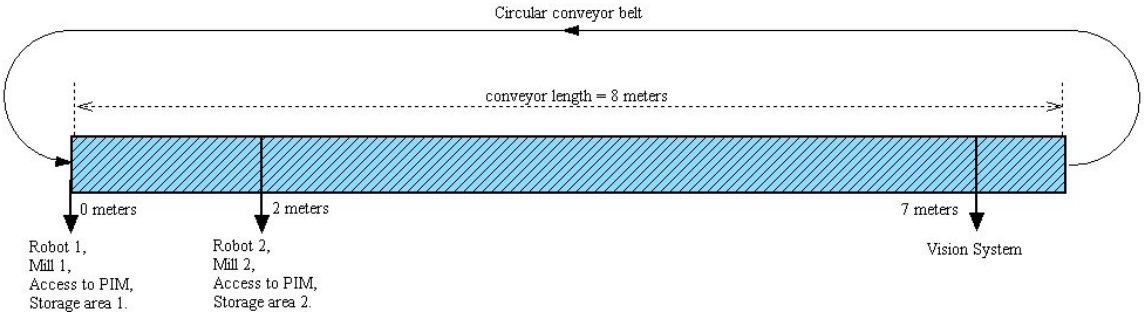

Figure 4.3. The relative positions of equipment.

4.3 Control Reconfiguration (System Level Results)

4.3.1 Object Model

The object-based hybrid model structure for the testbed is described in this section. The input output relations between the objects are shown in Figure 4.4. The conveyor has proximity sensors that indicate the palette is at position where the robot should pick up the product or when an image should be taken by the camera. The robot initiates plastic injection and milling operations when the robot has finished the appropriate placing operation.

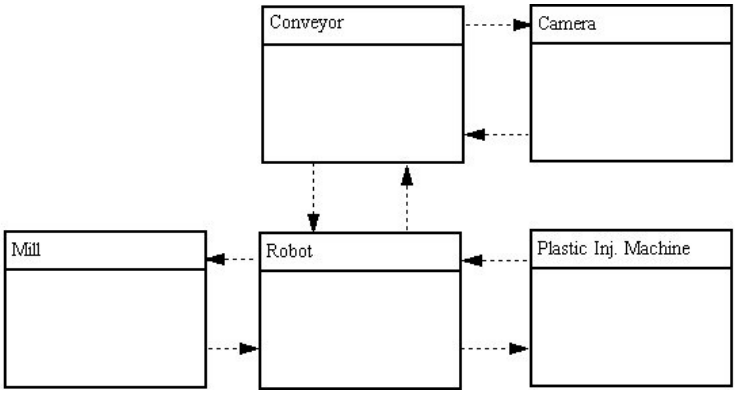

Figure 4.4. Subsystem input/output relationships.

Object submodels for each of these subsystems were created. These object submodels have object attributes, state attributes, and operations.

The conveyor object model is divided into the conveyor motor, conveyor track, and palettes that lay on the track are shown in Figure 4.5. The major operations of the conveyor are:

1. The conveyor motor speed is set.

2. The conveyor motor is turned on.
3. The track moves due to the conveyor motor and gears.
4. The palette moves due to friction between the track and the palette.
5. Proximity sensors detect when a palette is either near the camera or robot.
6. Palette stop actuators can be induced to stop the palette at certain positions.

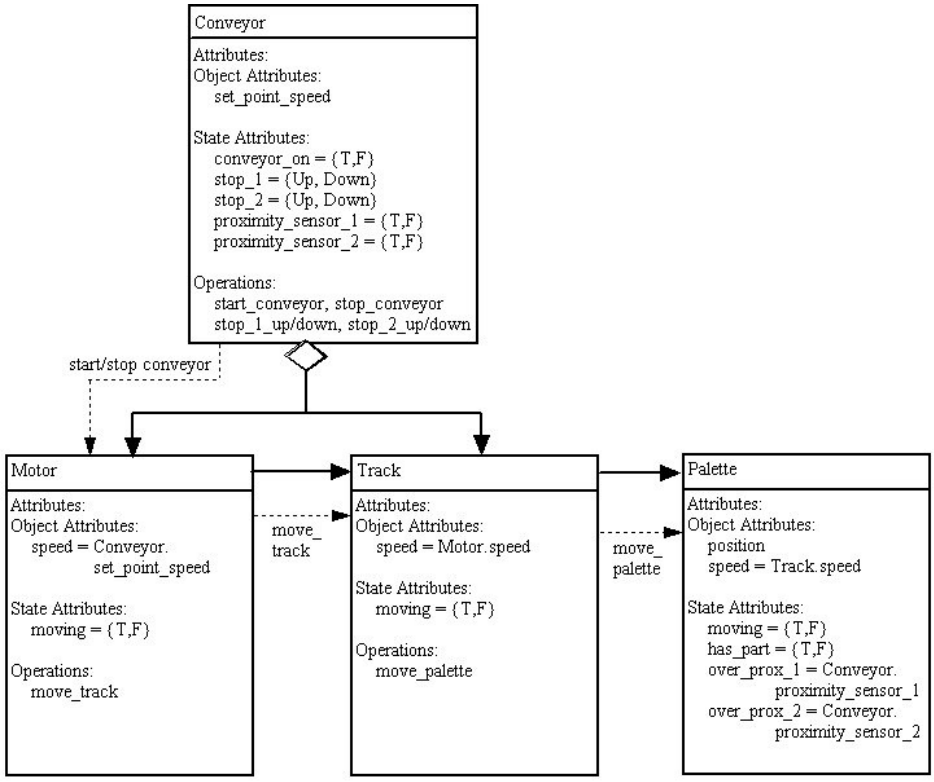

Figure 4.5. Conveyor object submodel.

The robot object model divides the robot into the separate motors (links) as shown in Figure 4.6. Each link is controlled by a motor which determines the position of the end effector. The major operations of the robot are:

1. Set the robot speed.
2. The robot can be moved to different positions by commanding each link to go to different angles.

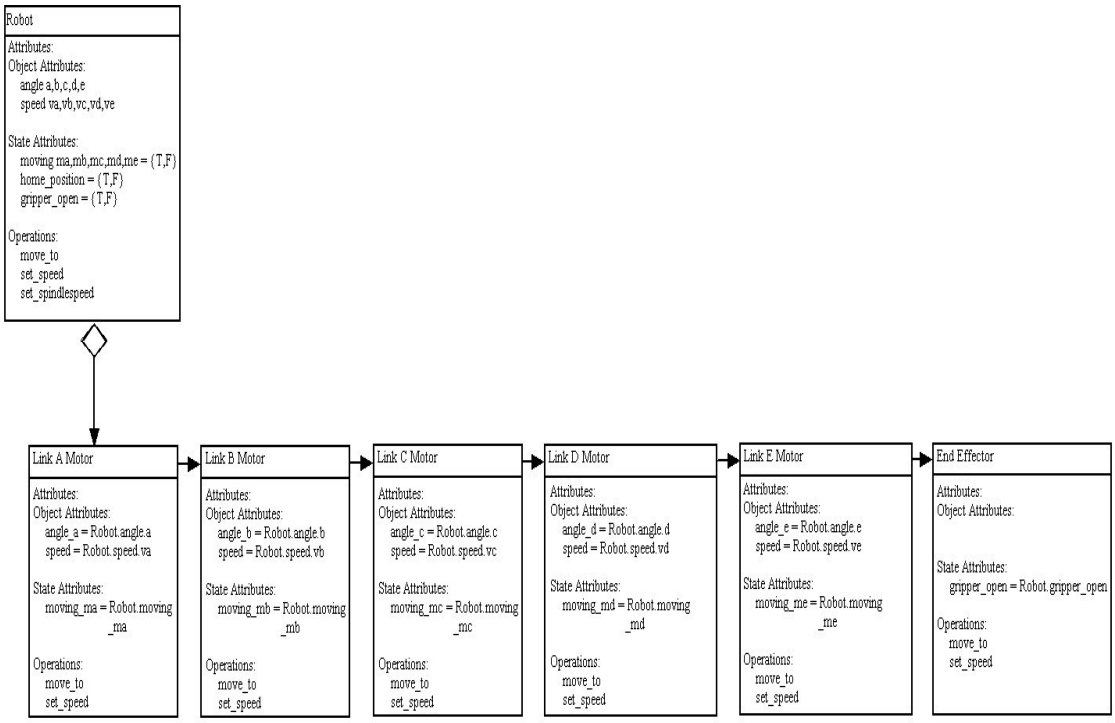

Figure 4.6. Robot object submodel.

The mill object submodel is divided into x, y, and z axis motors and the spindle as shown in Figure 4.7. The x and y motors control the horizontal positioning of the part underneath the mill. The z-axis motor controls the positioning of the spindle relative to the part. The spindle is speed controlled for milling of the part. The major operations of the mill are:

1. Set the feed rate and spindle speed.

2. Move the part to different positions.

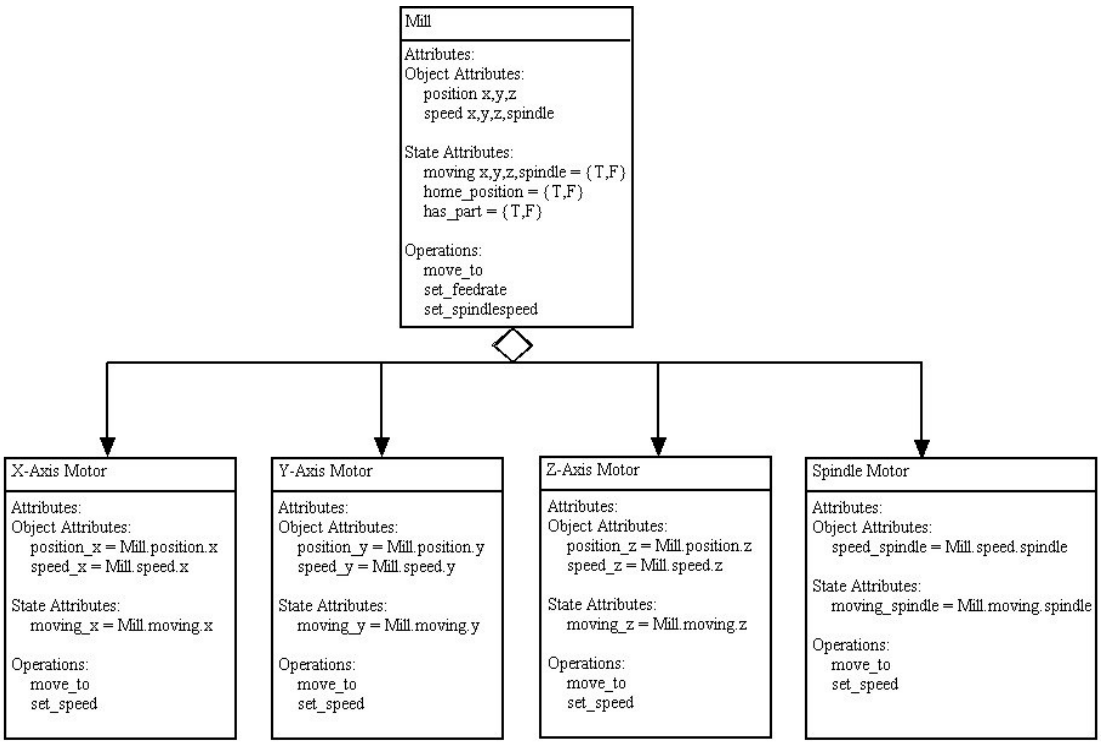

Figure 4.7. Mill object model.

The vision system object submodel in Figure 4.8 represents the camera taking an image of the part when the operation, take_image, is executed.

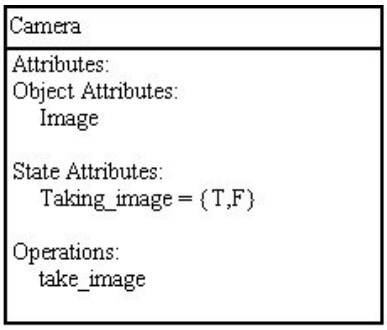

Figure 4.8. Vision system object model.

.A reduced object submodel is used to represent the plastic injection molding machine as shown in Figure 4.9. The object attributes represent the user input settings of the plastic injection molding machine. The product quality output is the estimated features of the product due to the current settings. The operations associated with this machine are simply start and stop the injection molding process.

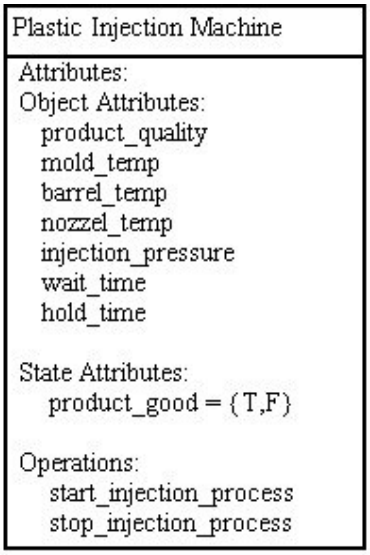

Figure 4.9. Plastic injection machine object model.

4.3.2 System Dynamics.

The control reconfiguration results are provided in this section. The supervisory Petri net for the modeled system is shown in Figure 4.10. There are a total of 8 different paths a part can take through the system when no failures are present. If a failure occurs that prevents a subsystem from being used (such as robot 1), then a part can be rerouted away from the faulty system by selecting another functional path.

The robot and conveyor are the only systems modeled with time-driven dynamics. The robot makes an excellent choice because motors control many subsystems in a manufacturing process. The robot time-driven dynamics are described by the following equation:

$$M(q)\ddot{q} + C(q,\dot{q}) + F(\dot{q}) + G(q) = \tau \tag{4.1}$$

where, $q \in R^n$ is a vector of robot arm joint angles, $M \in R^{nxn}$ is the inertia matrix, $C \in R^n$ is a vector representing the Coriolis and centrifugal force, $F \in R^n$ describes the viscous and Coulomb friction, $G(q) \in R^n$ is the gravity loading vector, and $\tau \in R^n$ are the joint torques. The robot is controlled using a PID controller described the following equation:

$$\tau(t) = K_p e(t) + K_I \int_{-\infty}^{t} e(\eta) d\eta + K_D \frac{de(t)}{dt} \tag{4.2}$$

where, $K_p \in R^{n\times n}, K_I \in R^{n\times n}, K_D \in R^{n\times n}$ are the proportional, integral, and derivative gain matrices, and $e(t) = q_T(t) - q(t)$ is the tracking error between the joint angles trajectory, $q_T(t)$, and the current joint angles. The robot is simulated using the Robotics Toolbox for MATLAB [21] that provides PUMA560 robot with 6 degrees of freedom that are used in the following simulations.

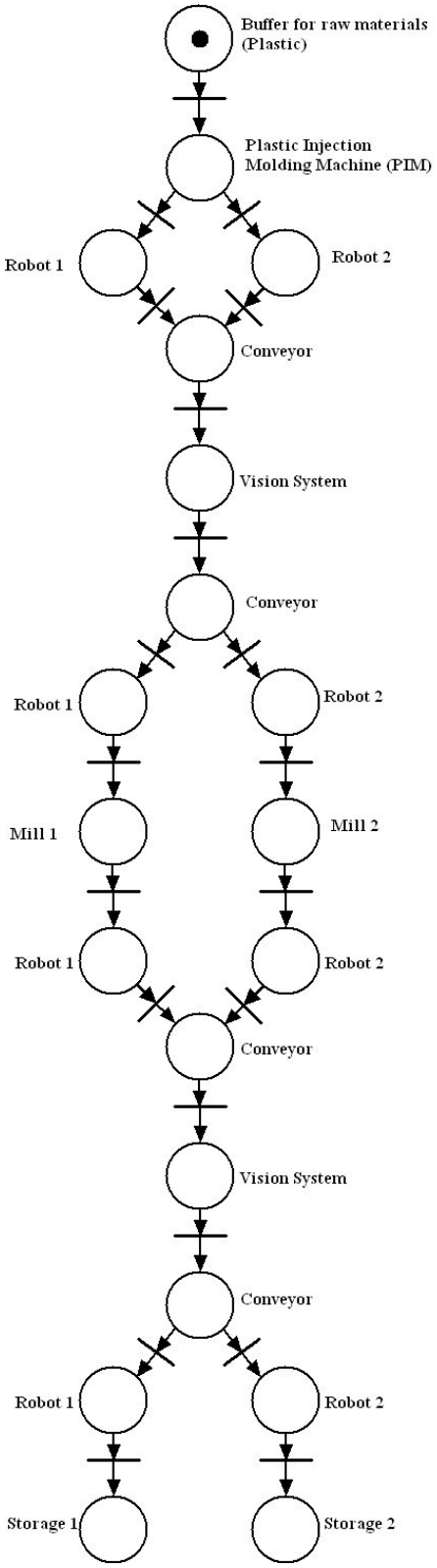

Figure 4.10. The supervisory Petri net.

The robot event-driven dynamics are shown in Figure 4.11. The robot moves from a current position state to a destination state through the "Goto" places. Once it reaches the required state, a Petri net, transition fires and the current "Goto" operation is completed. The "Break" place necessary for holding the state of the net when the robot has completed the operation. Currently the simulation assumes that a brake is enforced on all joints once the destination is achieved. The "Pick Part" and "Place Part" operations are not described by time-driven dynamics, because the operations on the current testbed do not require complex placement of parts.

The conveyor belt is a circular belt that is in continual operation during the process. A simplified model of the time-driven dynamics are expressed in the equation:

$$\dot{x} = r \tag{4.3}$$

where, r is the rate (meters/second) of the conveyor belt , and x is the position of a part (in meters) while on the conveyor belt and not stopped by a palette-stop. A modulus function (x mod 8) is performed on the position since this conveyor belt is circular.

The event-driven dynamics of the conveyor belt are described in the Petri net in Figure 4.12. The part is either in motion or not. The event that a part arrives at its destination fires the transition to move to the "part stopped" place and an event that the part is placed on the conveyor starts fires the transition to move to the "part moving" place.

The milling operation is simulated as a waiting time of 3 minutes and the visual inspection process is assumed to take 3 seconds. The plastic injection molding process is not modeled.

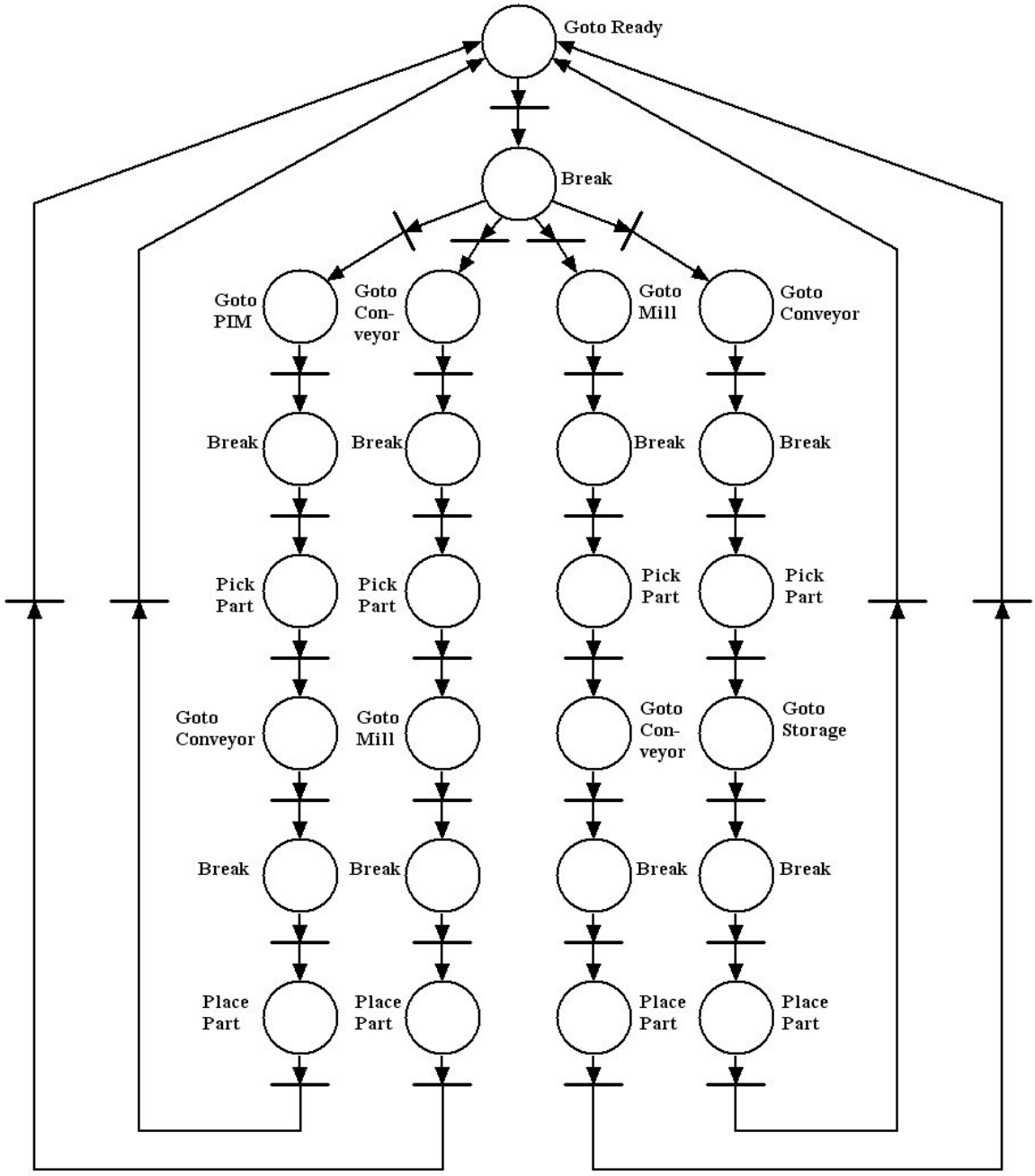

Figure 4.11. Petri net describing Robot event-driven dynamics.

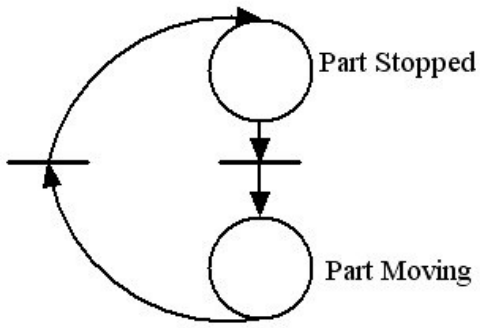

Figure 4.12. Petri net describing conveyor event-driven dynamics.

4.3.3 Simulations with no Failures Present

The following graphs show the simulations for the no failure case and the PID controller for the path that only utilizes robot 1, mill 1, storage 1, and the conveyor belt. Figures 4.13-4.28 are shown in sequence that the operations occur, dictated by the supervisory and event-driven Petri nets. Figures 4.13, 4.14, and 4.15 demonstrate the robot 1 operation of taking the raw materials from the plastic injection molding machine (PIM), placing it on the conveyor for later visual inspection, and returning to the ready position. Figures 4.10 and 4.11 show the conveyor moving the part to the visual inspection system and returning the part to robot 1. Figures 4.29 to 4.32 show the states of the supervisory, conveyor, robot 1 and robot 2 Petri net states. The Petri net state of robot 2 is constant since it is not used in this example.

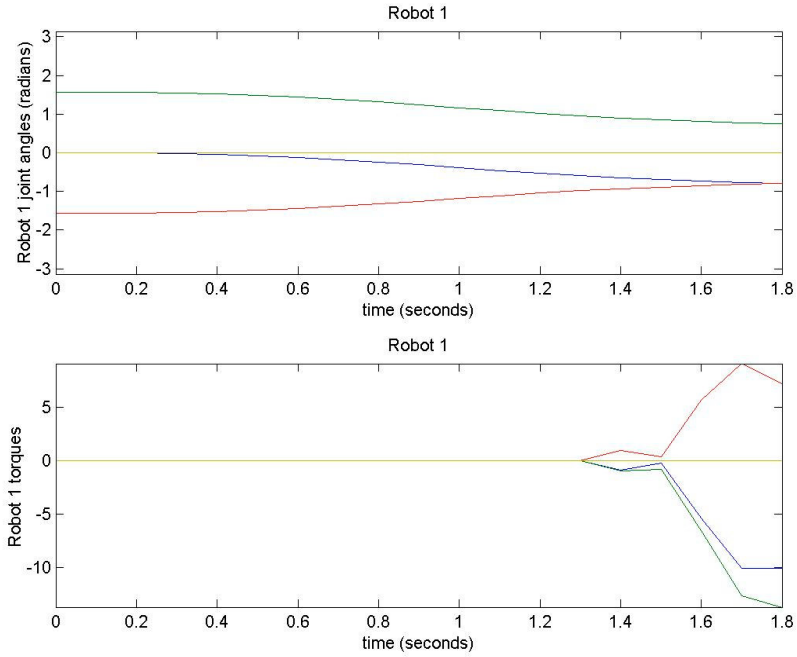

Figure 4.13. Robot 1 moving from ready state to PIM.

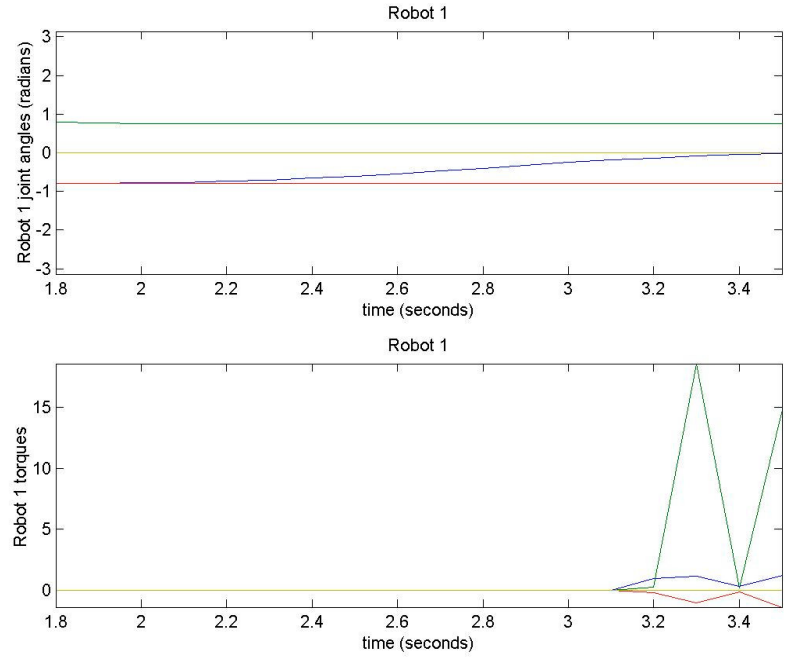

Figure 4.14. Robot 1 moving from PIM to conveyor.

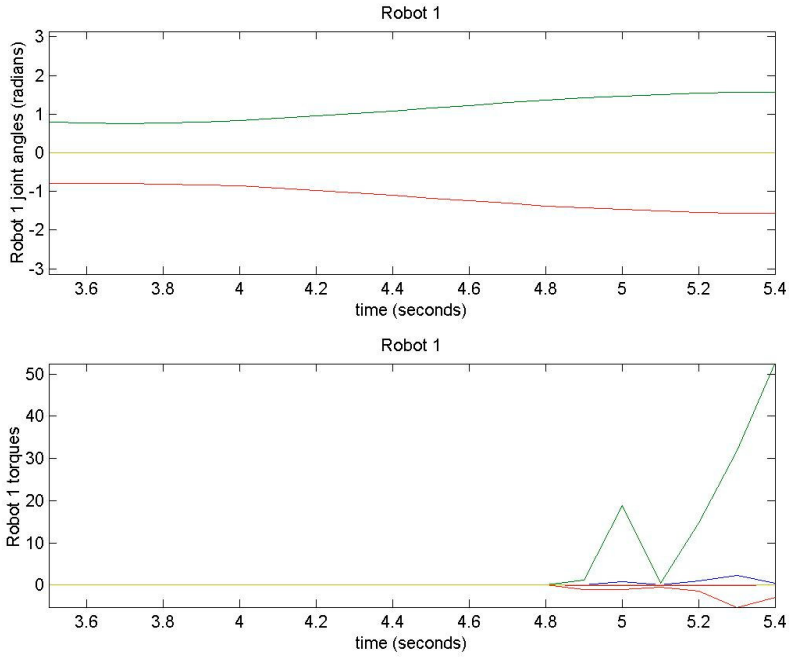

Figure 4.15. Robot 1 moving from conveyor back to ready state.

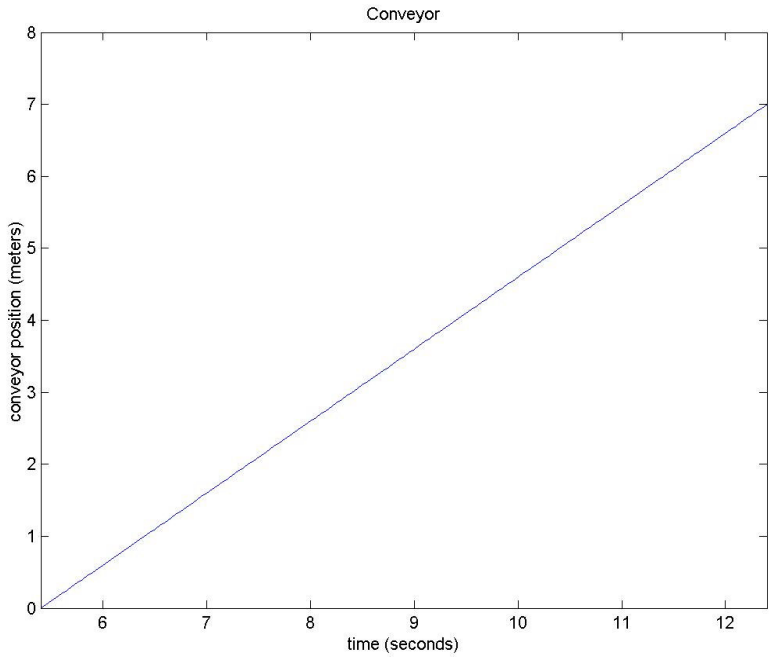

Figure 4.16. Conveyor moving part to vision system.

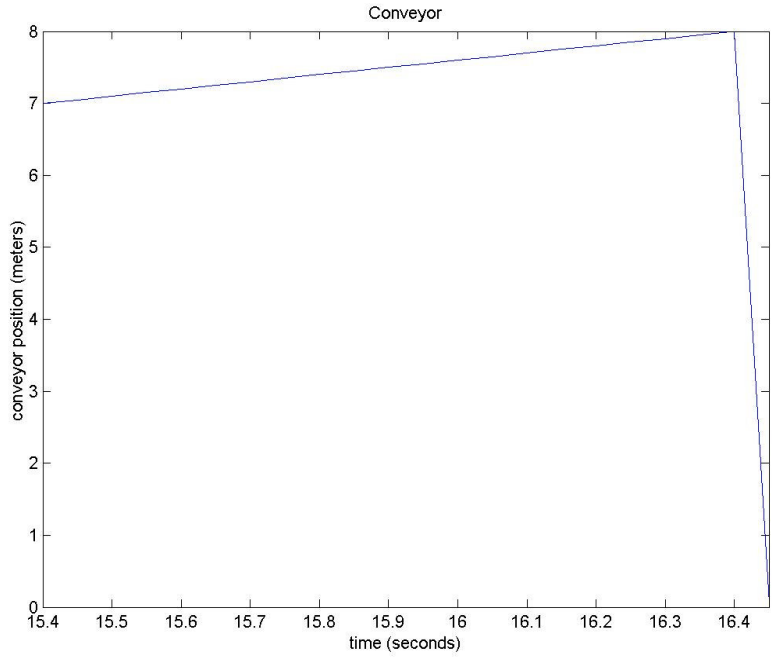

Figure 4.17. Conveyor moving part to back to robot 1.

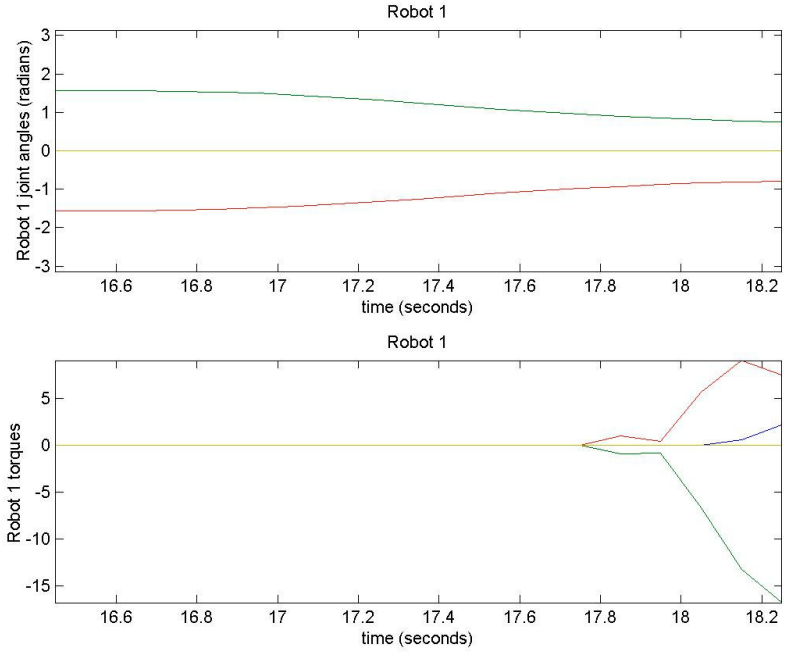

Figure 4.18. Robot 1 moving from ready state to conveyor.

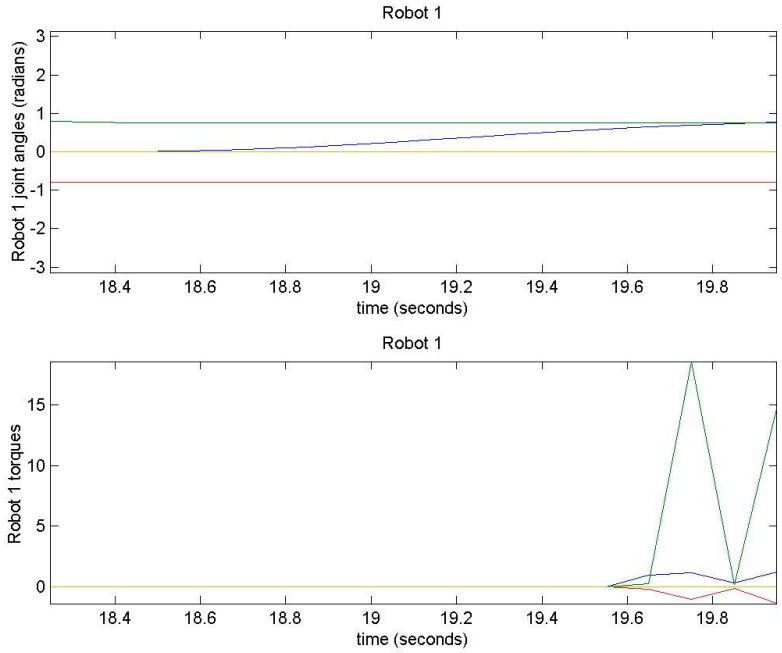

Figure 4.19. Robot 1 moving from conveyor to mill 1.

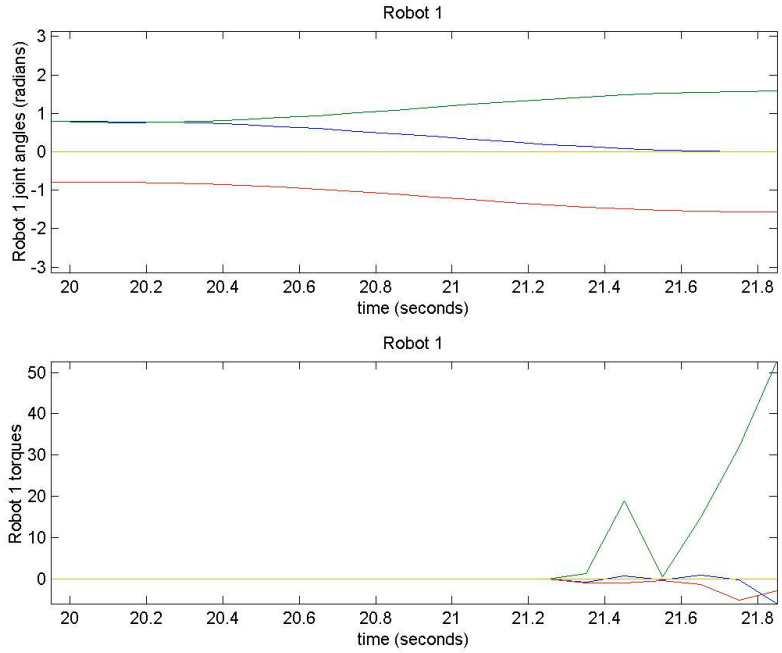

Figure 4.20. Robot 1 moving from mill back to ready state.

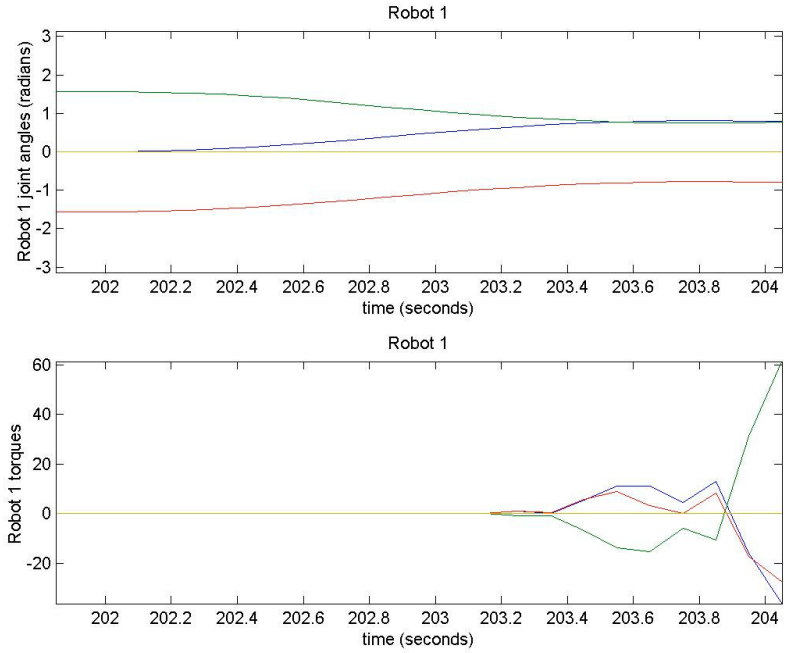

Figure 4.21. Robot 1 moving from ready state to mill 1 (after milling operation)

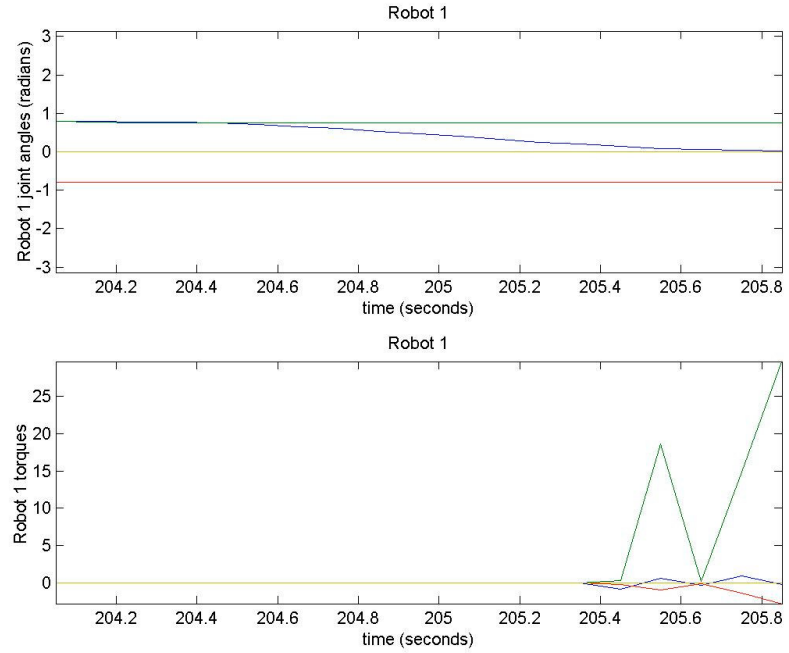

Figure 4.22. Robot 1 moving from mill 1 to conveyor.

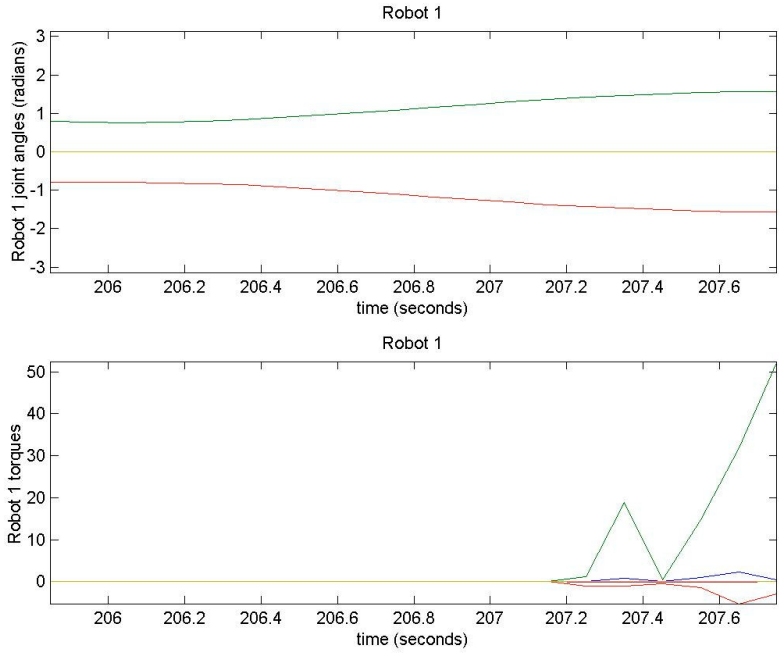

Figure 4.23. Robot 1 moving from conveyor to ready state.

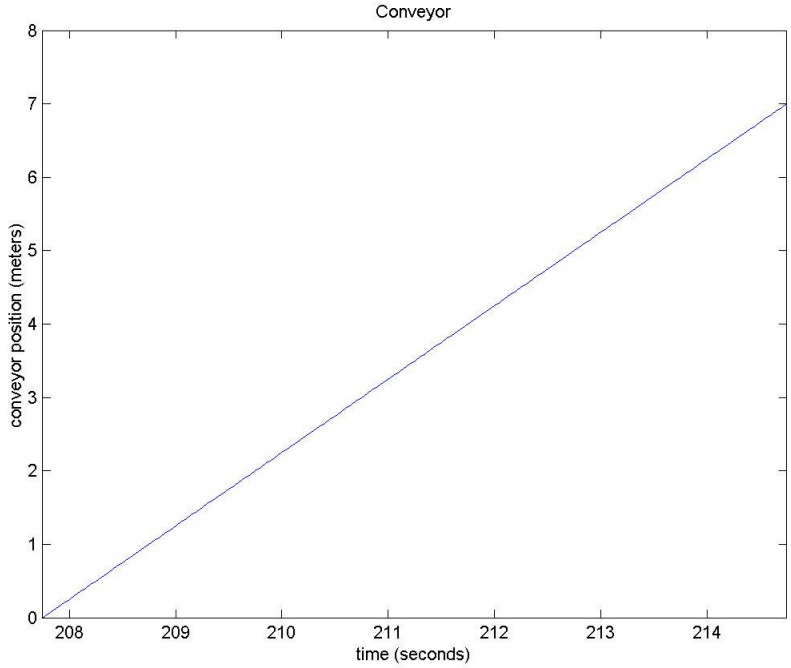

Figure 4.24. Conveyor moving part to visual system.

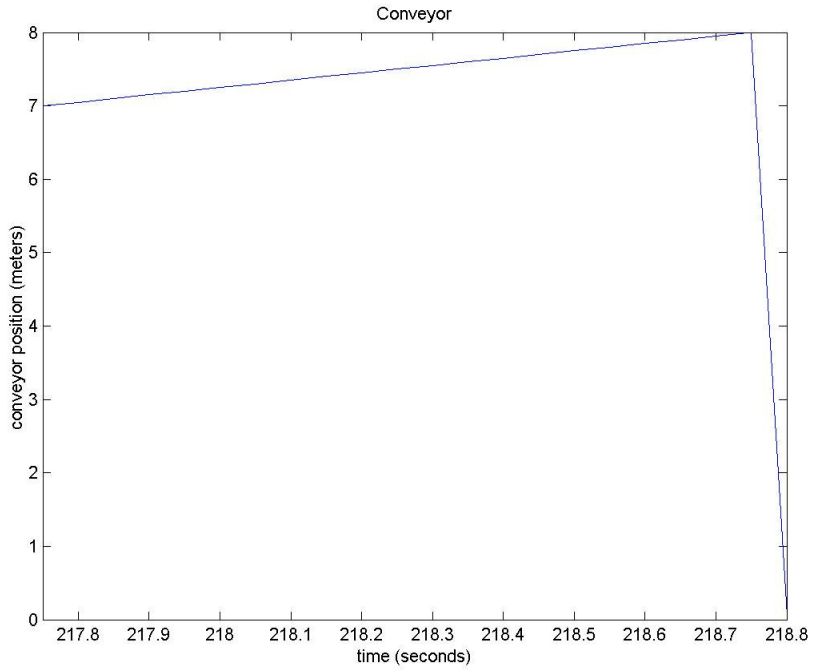

Figure 4.25. Conveyor moving part back to Robot 1.

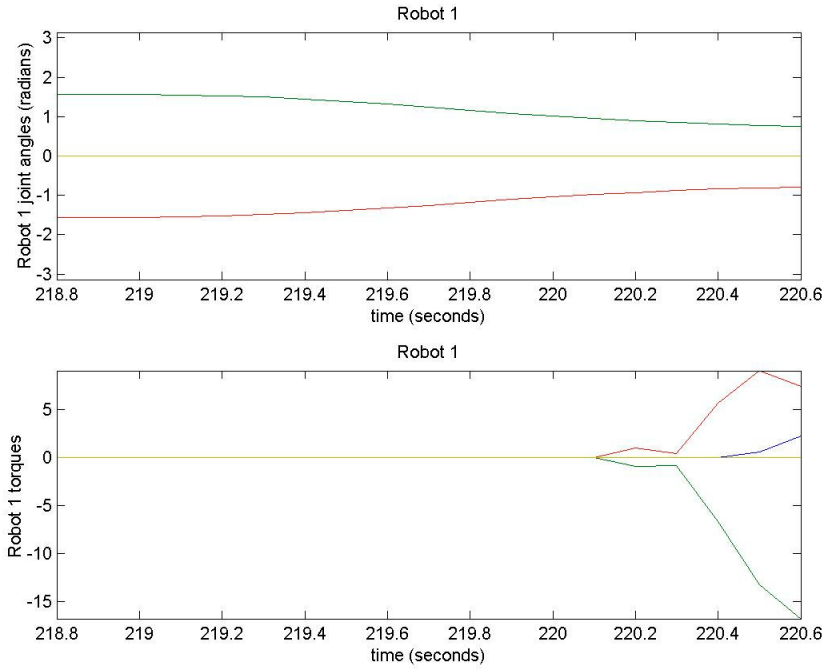

Figure 4.26. Robot moving from ready state to conveyor.

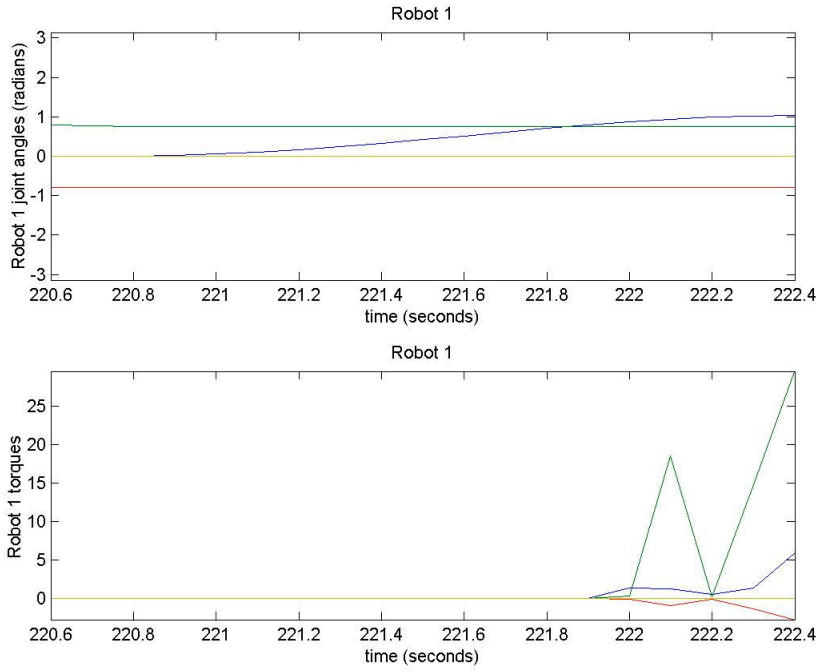

Figure 4.27. Robot moving from conveyor to storage 1.

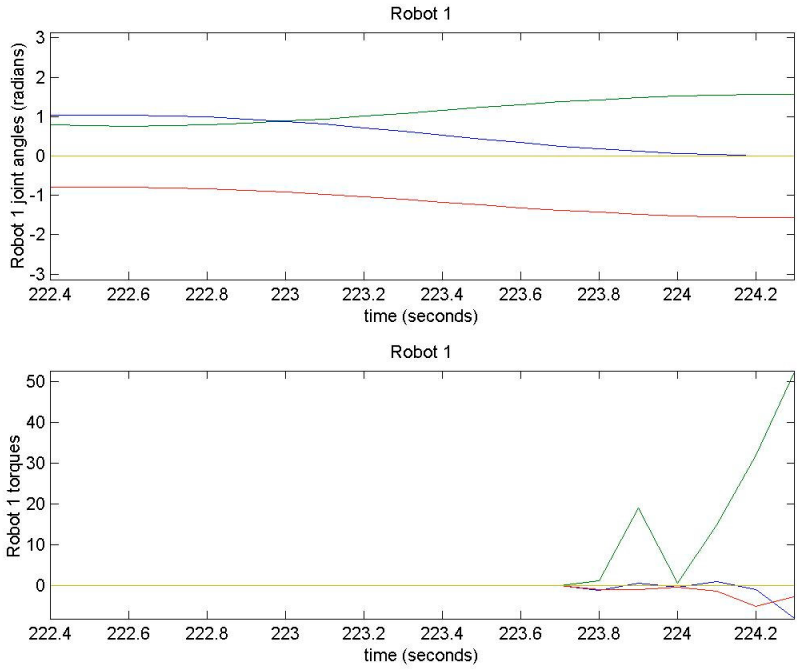

Figure 4.28. Robot moving from storage 1 to ready state.

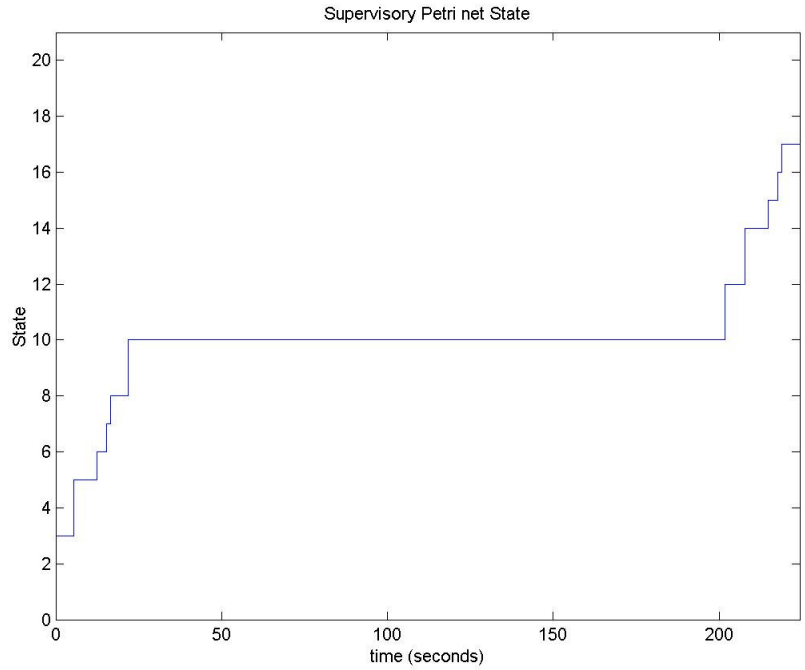

Figure 4.29. Supervisory Petri net state.

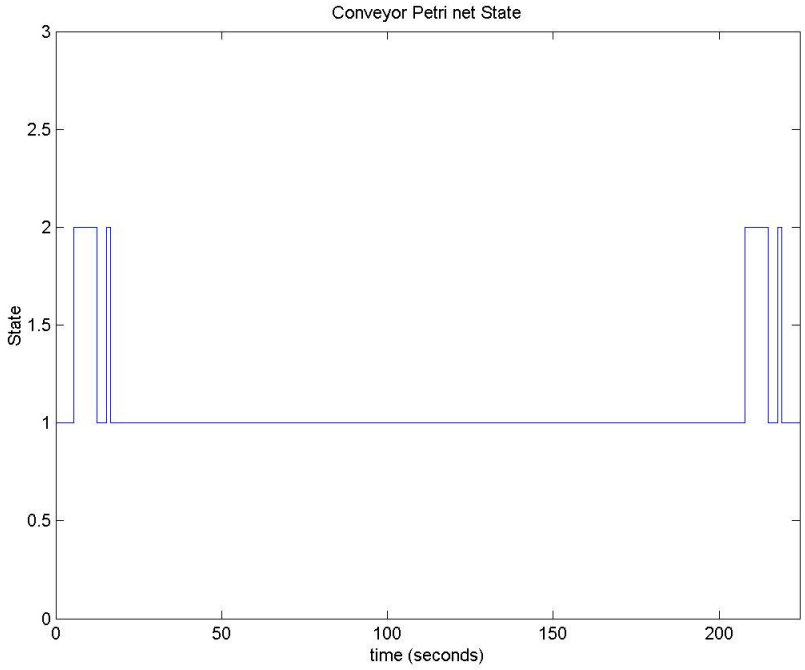

Figure 4.30. Conveyor Petri net state.

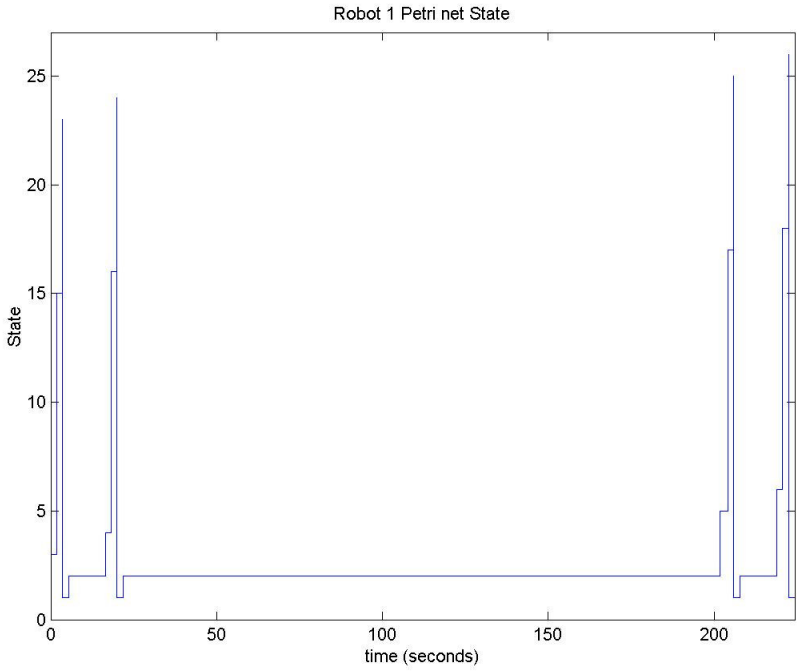

Figure 4.31. Robot 1 Petri net state.

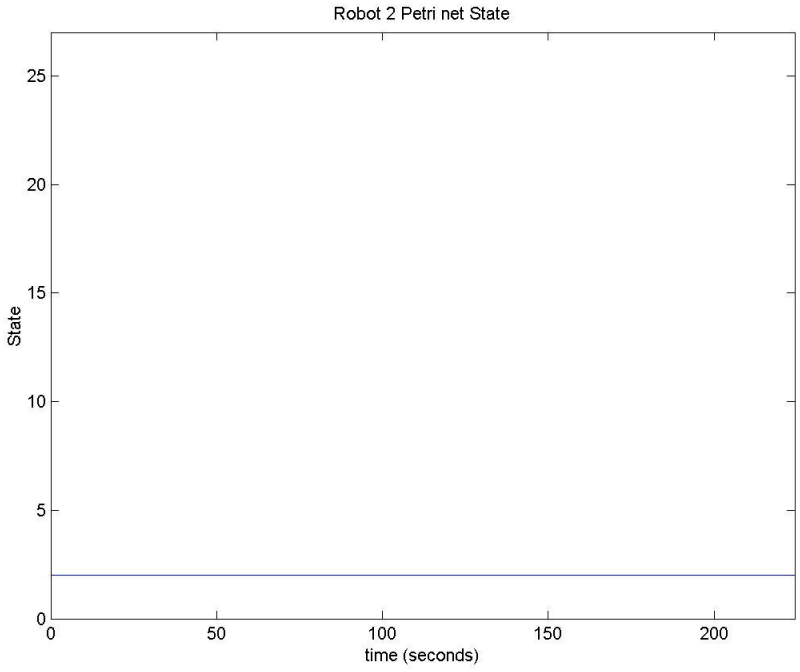

Figure 4.32. Robot 2 Petri net state.

Simulations were performed for all possible paths and the total time recorded in Table 1. Paths 5 and 7 yield the fastest times because these paths choose robot 2 to pick the part from the plastic injection molding machine and it is closer to the visual inspection system. Paths 2 and 4 yield slowest times because the parts travel a longer distance to get to the storage facilities by choosing robot 2. Robot 1 and robot 2 travel different distances to get to various locations as well, but these differences seem to not affect the time. This is because the trajectory path was designed to travel a distance in a specified time. However, in many cases, commercial robots provide a speed setting and if it were constant there would be some differences in time to completion for using different robots that traverse different distances (although it might be small).

Table 1. Time for single part completion for different paths.

	Path 1	Path 2	Path 3	Path 4	Path 5	Path 6	Path 7	Path 8
Time (sec)	224.3	226.3	224.3	226.3	222.2	224.2	222.2	224.2

4.3.4 Simulations with Failures Present

In this section, we consider failures for a linear type system example with the adaptive controller. The simulated system is represented by the equations:

$$\dot{x}_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + b_{11}u_1 \tag{4.4}$$

$$\dot{x}_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + b_{22}u_2 \tag{4.5}$$

$$\dot{x}_3 = a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + b_{33}u_3 \tag{4.6}$$

This type of system can have redundancies in the controller and states when appropriate constants a and b are available. The simulated system has the constants:

$$\dot{x}_1 = 0.3x_1 + 0.3x_2 + 0.3x_3 + 0.3u_1 \tag{4.7}$$

$$\dot{x}_2 = 0.3x_1 - 0.3x_2 + 0.3x_3 + 0.3u_2 \tag{4.8}$$

$$\dot{x}_3 = 0.3x_2 + 0.3u_3 \tag{4.9}$$

The goal of the example system is to track a constant reference signal so that $x_1 = 1$. A proportional-integral (PI) controller is used to perform this action in the non-failure case. Figure 4.33. shows the simulated results where the blue plot is state, x_1. .

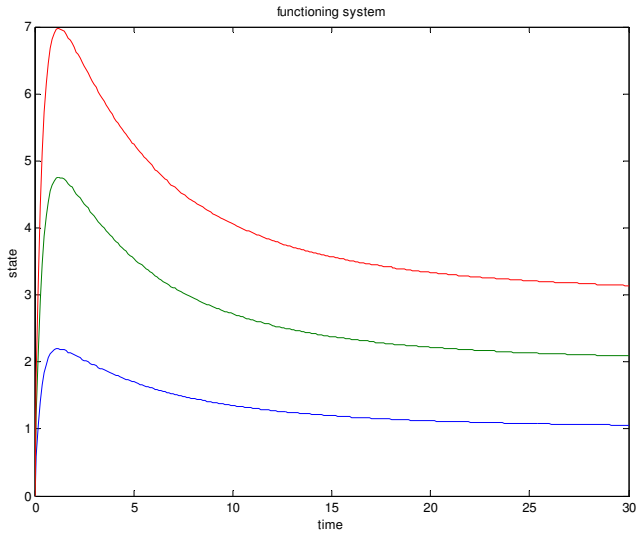

Figure 4.33. Simulation, no failure case.

The error between the reference signal, $r = 1$ and the state variable, x_1 is shown in Figure 4.34.

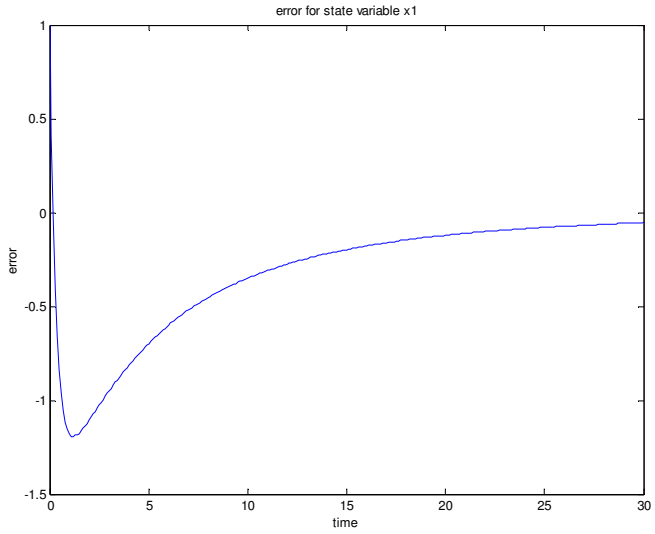

Figure 4.34. State error, no failure case.

The results show that the system will successfully reach $x_1 = 1$ as $t \to \infty$.

Now we introduce a failure in the system where some system parameters are changed. The faulty subsystem is represented by the equations:

$$\dot{x}_1 = 5x_1 + 0.3x_2 + 0.3x_3 + 0.3u_1 \tag{4.10}$$

$$\dot{x}_2 = 0 \tag{4.11}$$

$$\dot{x}_3 = 0.3x_2 + 0.3u_3 \tag{4.12}$$

This type of failure causes state x_2 to stay constant and a parameter for x_1 in Equation 4.10 is changed from 0.3 to 5. Such failures as setting one state constant are similar to real world applications such as a robot joint failure. Figure 4.35 shows the results with the PI controller when this failure is induced at time, $t = 3$ and the state error is shown in Figure 4.36.

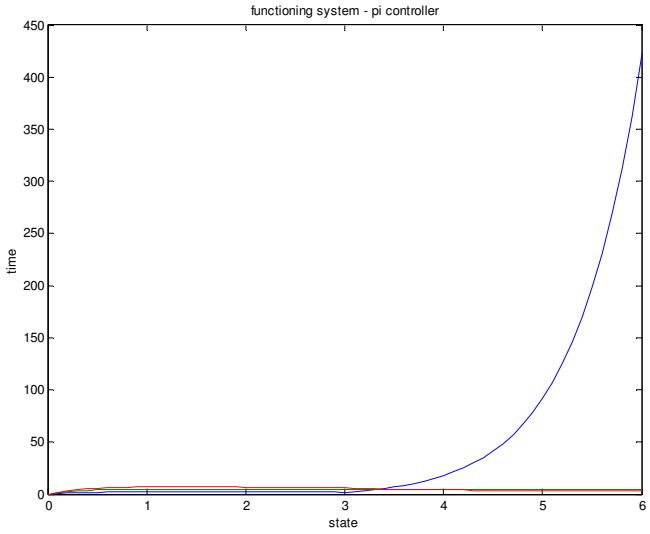

Figure 4.35. PI controller, failure at t=3.

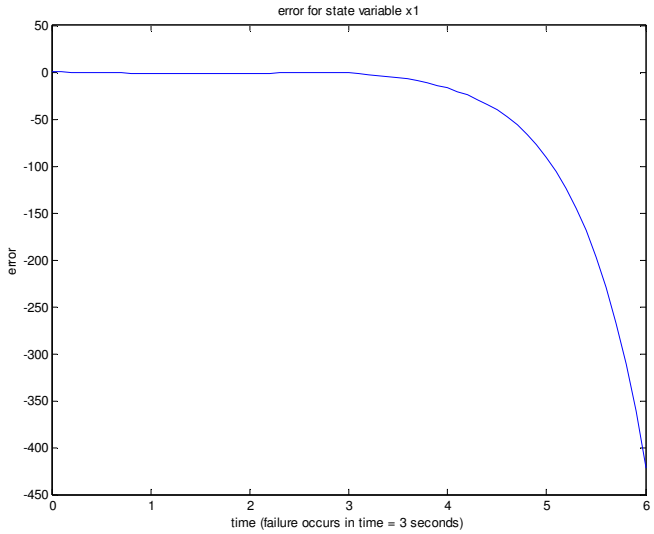

Figure 4.36. State error, failure at t=3.

Clearly, the PI controller no longer stabilizes the system. Figures 4.37 and 4.38 show the results for the adaptive controller.

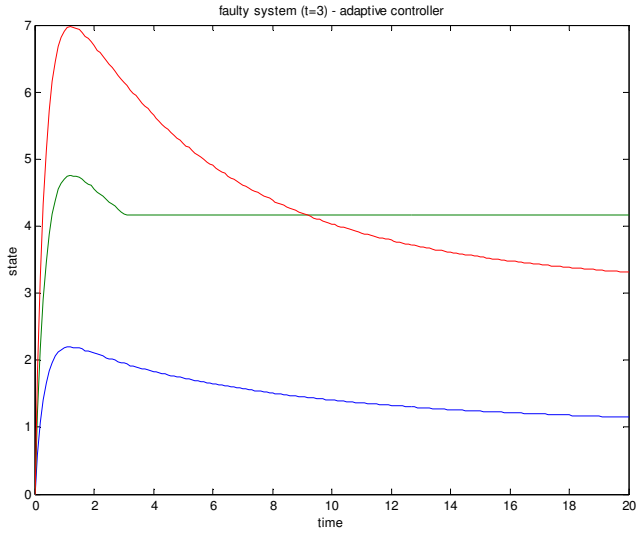

Figure 4.37. Adaptive controller results, failure at t=3.

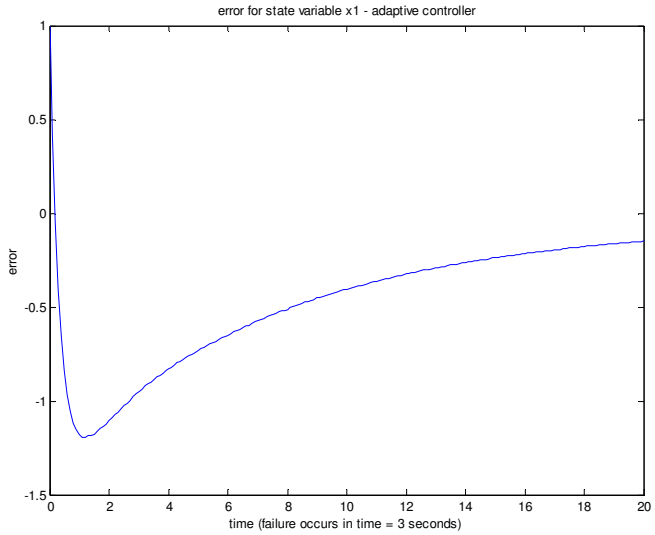

Figure 4.38. State error, failure at t=3.

The adaptive controller is able to compensate for the failure and achieve its goal.

4.4 Control Reconfiguration (Product Level)

4.4.1 The Product

Plastic parts will be created by a vertical, plastic injection molding machine from Morgan Press. Polystyrene is the type of plastic that is being used to create the plastic parts since it can be easily obtained and machined. The mold is in a shape of a small block of dimensions of size 1.5" X 1.5" X 0.5" with a stem down the center. The stem is from the sprue of the molding process and will be shaved off by a milling operation. The machining operation will consist of first milling out a rectangle 0.1" deep within the center of the part. The next operation will then mill out a circle 0.2' deep within the center of the part. Figure 4.39 shows an image of the product after all operations are performed. The critical parameters of the part are the quality of the plastic, the dimensions and positioning of the milled square and circle, and the quality of the milling operation on the part.

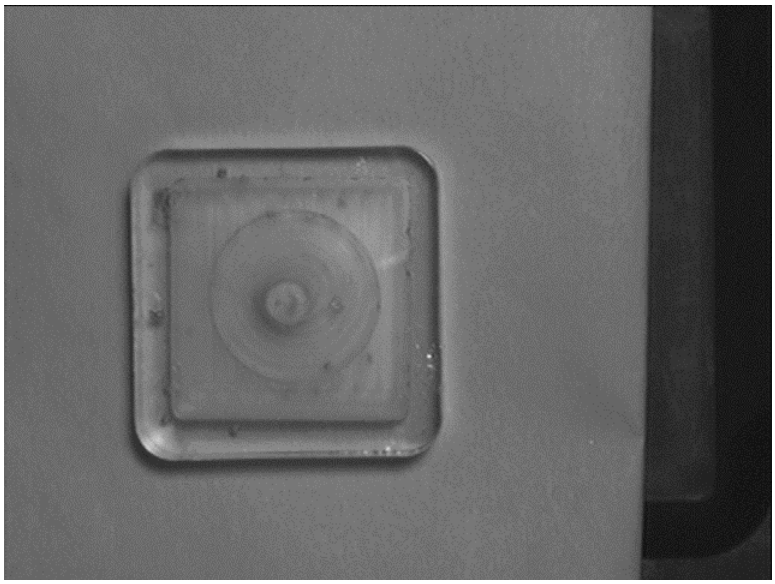

Figure 4.39. The product.

4.1.2 The Faults

In this section, the plastic injection molding process is described as well as the possible faults associated with each operation. The plastic injection molding process can be reduced to the following major steps:

1. Plastic beads are inserted into the machine barrel for melting.
2. Temperature set-points are adjusted for the barrel that holds the plastic, the nozzle where the plastic leaves the barrel, and the mold temperature. The temperatures are dependent on the type of plastic used.
3. Pressure set-points are set for the ram injecting the plastic through the barrel and the clamping tonnage of the mold.
4. After the correct temperatures are reached, the mold is clamped together.
5. Pressing the injection switch starts the injection process where the ram presses molten plastic in the barrel and then through the nozzle into the mold.

6. There is a wait time to make sure that a sufficient amount of plastic has been injected into the mold.
7. The injection process is stopped after completion of the wait time.
8. There is a hold time here to make sure that the plastic has cooled sufficiently.
9. The mold is unclamped after completion of the hold time.

There are many different faults associated with a plastic injection process. Here, we take a subset of these failures for analysis on the manufacturing testbed. The main faults under investigation are described below with possible causes and corrections through set-point operations.

1. No Fault Case (Figure 4.40)—This part is a good part. It has very few voids, no burns, no splay, low pull on the stem, and was completely filled with the plastic.

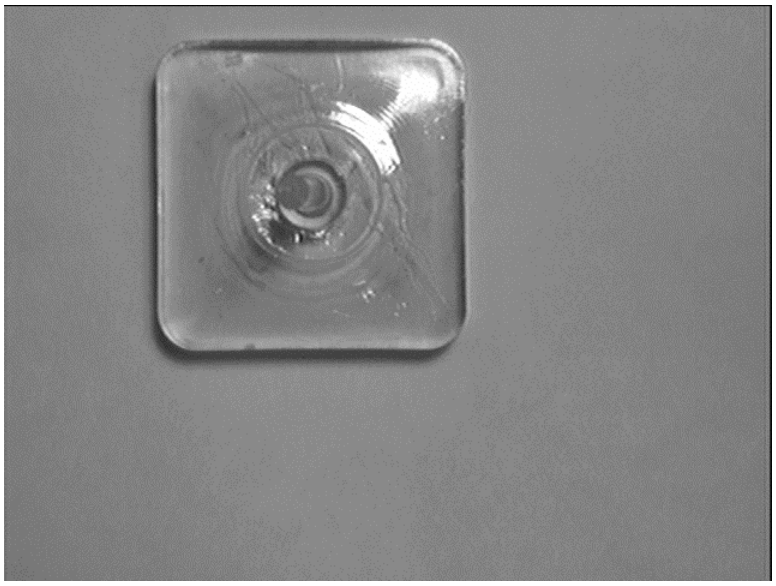

Figure 4.40. A good plastic part.

2. Voids (Figure 4.41)—This part has many pockets of trapped air within the part. The causes that are investigated for this fault are:
   - The air cannot escape the mold due to excessively high clamping pressure.
   - The injection was not at a high enough pressure to force air outside the molten part.
   - The wait time while injecting was not long enough to completely force air out of the mold.
   - The mold temperature was not high enough to keep the plastic flowing freely while in the mold.

Figure 4.41. A part that has too many voids (bubbles).

3. Burn (Figure 4.42)—This part has burned plastic inside or outside the part usually showing up as blacked or yellowed plastic. The causes that are investigated for this fault are:

- Temperature settings of the barrel temperature for the particular plastic type are too high.
- Temperature settings of the nozzle temperature for the particular plastic type are too high.
- The wait time (injection time) is too long causing the surface plastic near the stem of the part to burn due to excessive heating by the nozzle, which is in contact with the mold during the wait time.

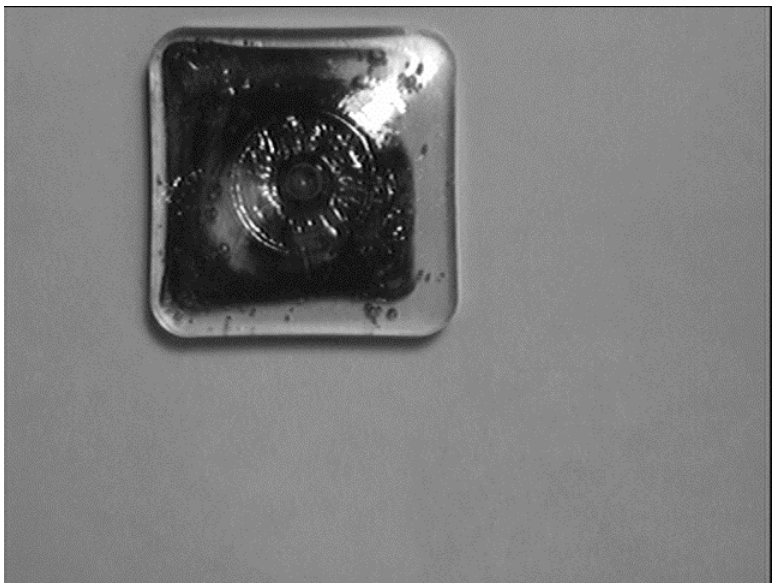

Figure 4.42. A part that has burned plastic material.

4. Short Shot (Figure 4.43)—This is the incomplete filling of the mold. The causes that are investigated for this fault are:
   - The wait time was not sufficient for the mold to completely fill.
   - The environmental conditions have changed and new set points are needed for the injection pressure, mold temperature, and nozzle temperature to help induce the flow of plastic. This can also occur from the shared use of an air supply.

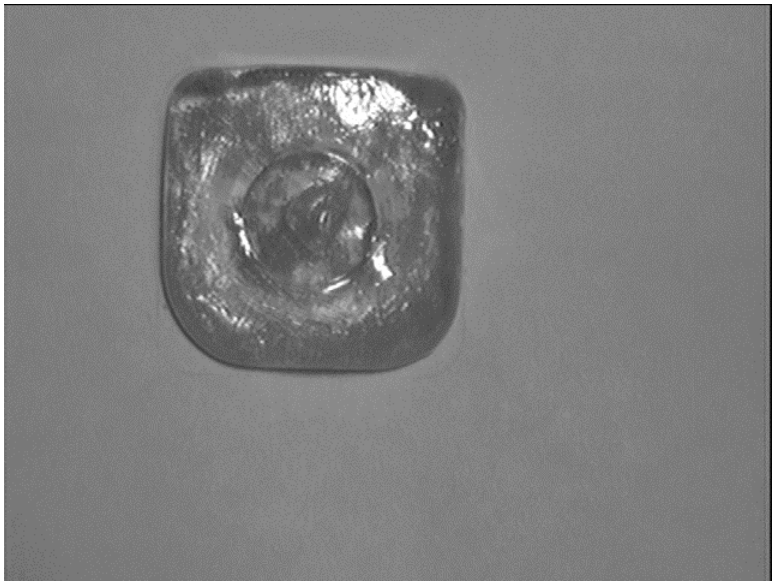

Figure 4.43. A part that is incompletely filled in the mold (a short shot).

The design of the part is the first step of the milling process. The part was designed with the packaged CAD/CAM software that came with the Light Machines mill. The CAD process is designs the type of stock used and the locations to be milled. The CAD editor possesses a variety of tools to define arcs, circles, squares, text characters, etc. The CAM software in Figure 4.44 defines the way the part should be cut (spiral, zigzag, outside or inside the boundary), the depth, and the feed rates. After the cuts have been defined, a NC program is generated.

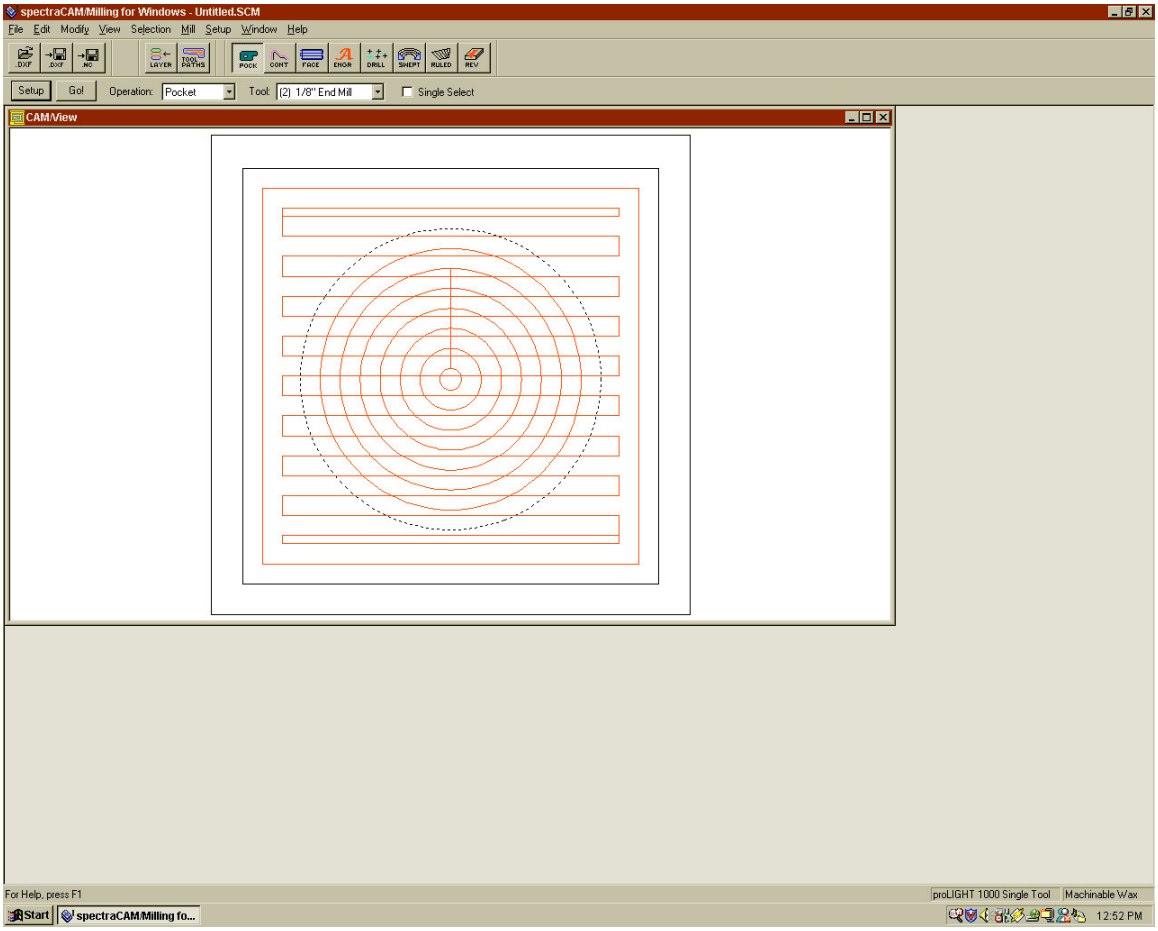

Figure 4.44. The CAM software.

The mill is then cuts the part using numerical control software (Figure 4.45) that communicates with the mill. The numerical control program is a line program with special codes defining the speed and feed rates, the types of cuts, and other external communication. The robot and mill communicate through external devices to signal when the raw part is ready to be placed in the mill by the robot and the milled part is ready to be picked up by the robot.

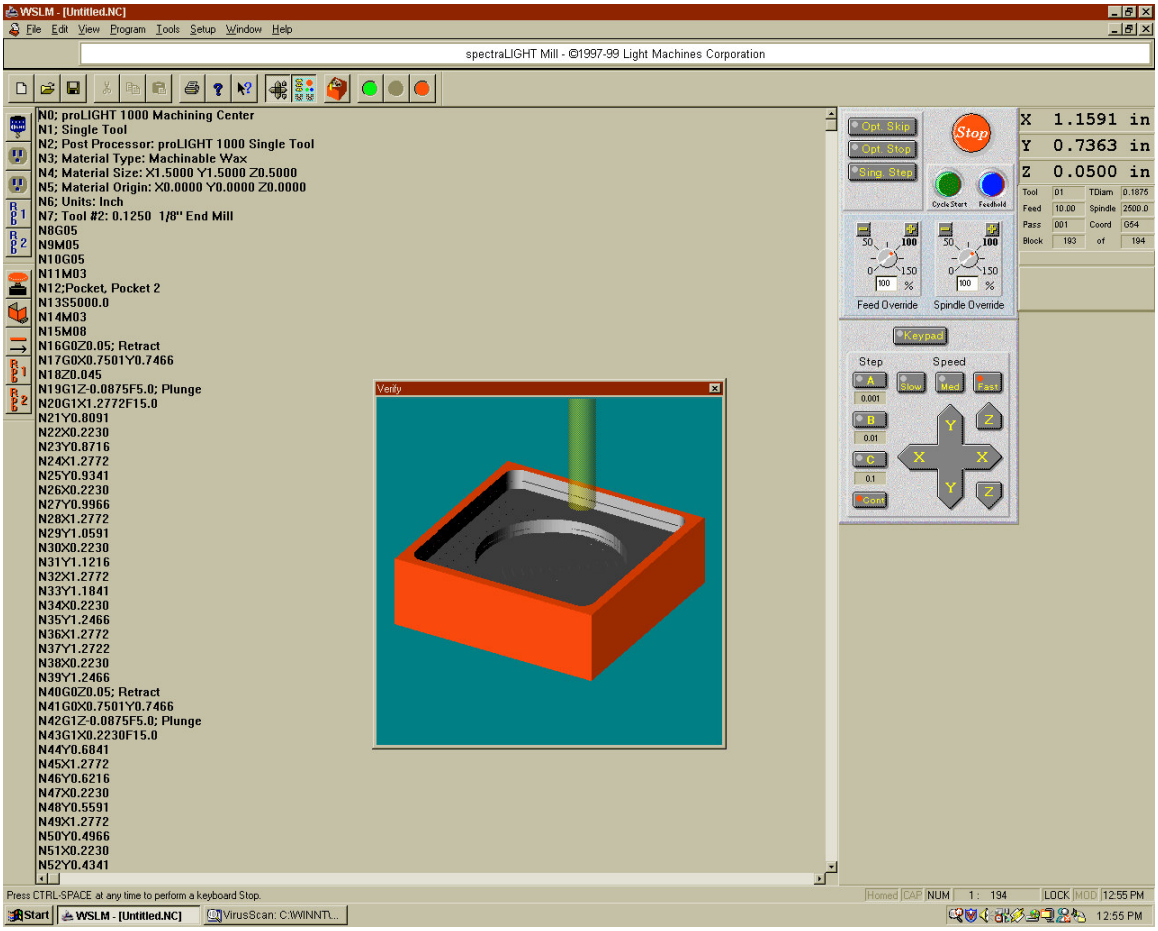

Figure 4.45. The numerical control software controller.

4.4.2 The Sensors and Features

There are various types of sensors that can be used to inspect a product. The primary sensor that is used for the testbed is a Pulnix camera that is used to inspect the quality of the product. A weight scale measurement is also used to detect lack of product. Proximity sensors are used to detect the position of the palette as it traverses the conveyor.

Features are extracted from the image by the following intuitive operations and measurements for the faults under consideration:

1. *Weight*: This is the measured weight of the product by a scale.
2. *Blackness*: This measures the darkness of the image by binary thresholding the image to values of 0 and 1 and then summing the pixel values.
3. *Whiteness*: This measures the whiteness of the image by first binary thresholding the image to values of 0 and 1 and then summing the pixel values.
4. *Area*: This feature measures the product area in the image by estimating the length and width of the product.

Performing binary thresholding at different levels and summing pixels from the image database randomly produces other product features. The feature selection process is performed using the genetic algorithm described in the previous chapter. Figure 4.46 shows the performance of the genetic algorithm. The chromosome with the smallest fitness value is chosen out of the number of trial runs. This tells us the features that should be used in the process. The x-axis represents each generation and the y-axis represents the fitness value of the generation.

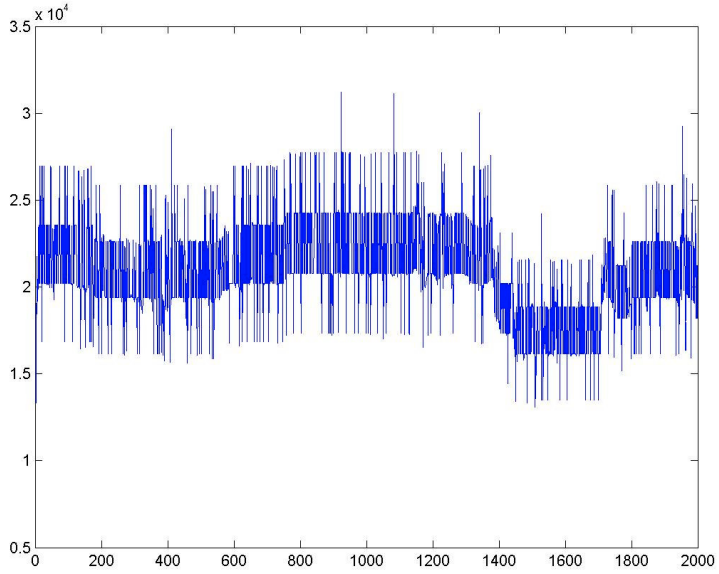

Figure 4.46. The fitness values of the defect feature selection process.

4.4.3 Product Reconfiguration Results

The control reconfiguration methodology is now performed on data gathered from the experiment testbed. Since the data was in short supply due to time constraints, new data was created based on the experimental data to yield more test results. The control reconfiguration consists of three steps:

1. Histograms are created for the feature values vs. the normalized frequency of occurrence.
2. The histograms are compared to the template histograms via histogram comparison features.
3. These histogram comparison features are then fed through an ANFIS structure to determine the set-point changes for the next run.

Figure 4.47 shows graphs of the distributions for 4 features of good parts and will be used as histogram templates. It used 10 points of data for each feature. The horizontal axis represent the feature value and the vertical axis represents the count. The features are upper left = block pixel area, upper right = blackness pixels, lower left = whiteness pixels, lower right = weight. These 4 features were selected from a set of 8 features from the feature selection process. The histograms were made with 61 bins within the range between –30 and 30 for each feature. This yields a bin width of 0.98 feature units for each graph. The number of bins will increase or the features will be multiplied by a scaling factor to enhance resolution.

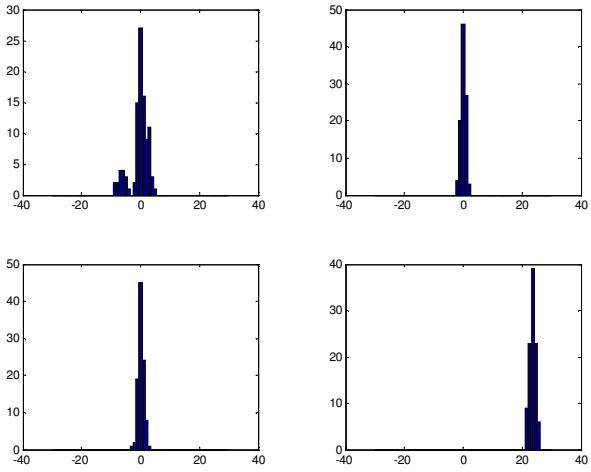

Figure 4.47. Template feature histograms.

Figure 4.48 shows histograms for a different set of good part data. The histograms are similar.

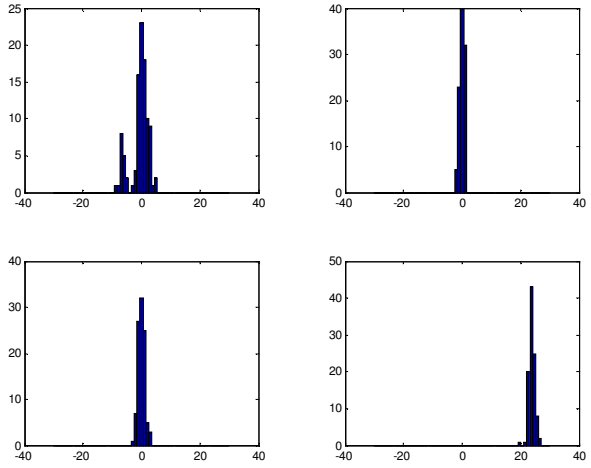

Figure 4.48. Feature histograms of good part data.

Figure 4.49 shows distributions of features for parts with bubbles. They look similar except for the block area feature.

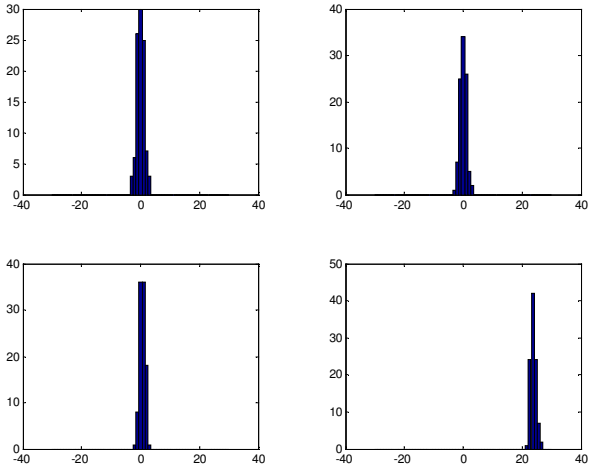

Figure 4.49. Parts with bubbles feature histograms.

Feature extraction is performed on the image data of the product and produces four features: area, whiteness, blackness, and weight. Frequency distributions of these features were produced for separate fault cases. Template frequency distributions for each feature were stored for a good product. For incoming data, each feature distribution is compared with the template frequency distributions by 11 distribution comparison metrics. Therefore there are $4 \times 11 = 44$ distribution comparison metrics. Feature selection was used to reduce the number of distribution metrics use in the comparison.

Comparisons were done on 5 cluster centers representing the 4 different product defects and the template distribution. The total fitness of the population on successive runs of the genetic algorithm on is shown in the Figure 4.50.

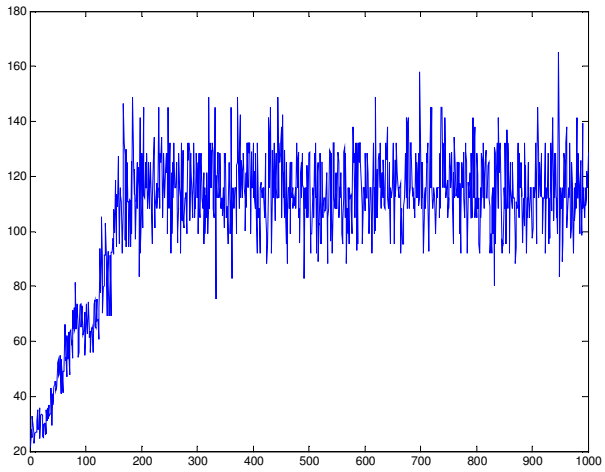

Figure 4.50. Feature selection for the feature histogram comparison metrics.

The results yield 6 metrics out of the original 11 metrics are used in the reconfiguration process.

Simulated results from the reconfiguration are now explained. Data was gathered for a bubble defect and then fed through the reconfiguration process. First, a trained, artificial neural network detected the bubble defect in the part and then the reconfiguration process was started. Histograms were created from incoming data and then comparisons were made with the template histograms through the comparison metrics. These results are then passed through the ANFIS reconfiguration routine and the set point adjustments were made until the product defect was no longer detected.

Figure 4.51 shows the resulting control commands when the bubble defect was detected. The pressure was increased to remove the bubbles and then leveled off as soon as the bubble defect was remedied.

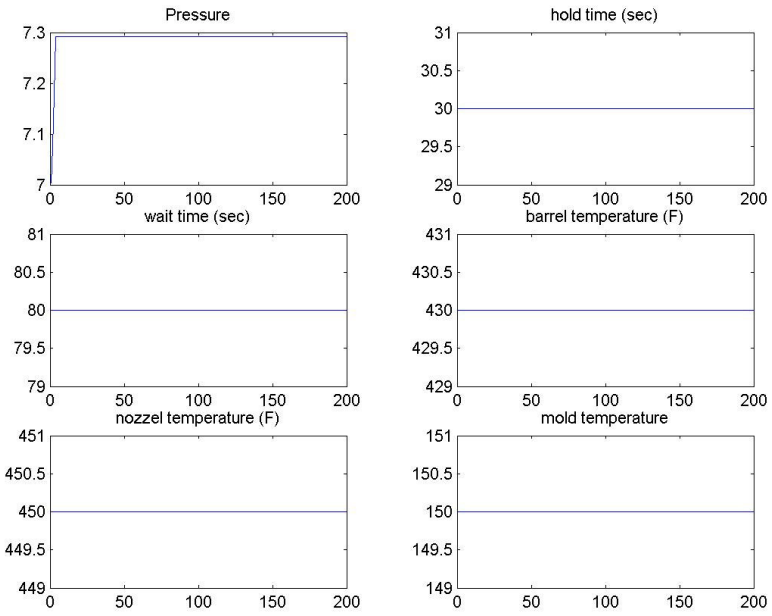

Figure 4.51. The control signals produced by the reconfiguration process.

A subset of the histogram comparison metric values are shown in Figure 4.52. The behavior suggests that there will be some residual error even when the defect has been removed. This residual error does not effect the process once the product defect is corrected since the reconfiguration process is shutdown after the set-points have been adjusted.

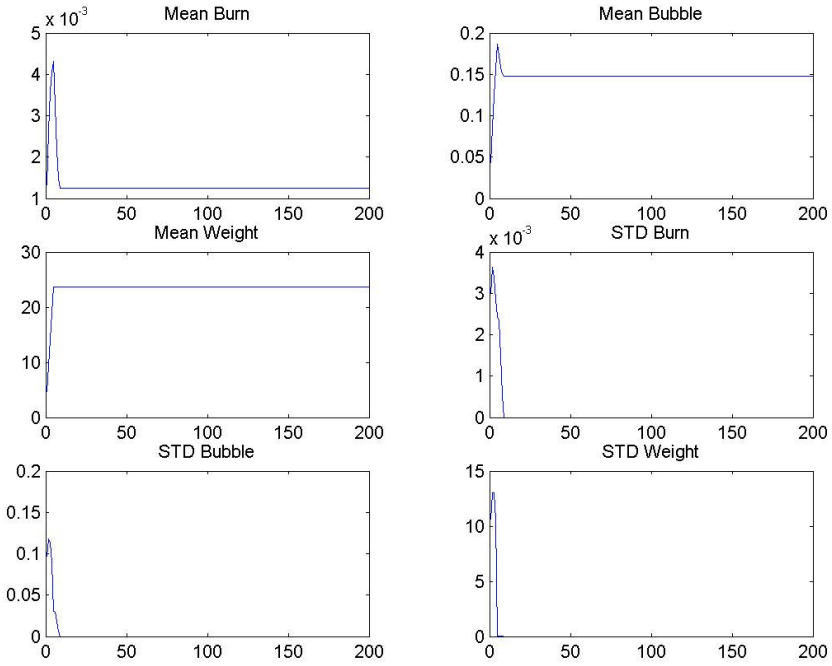

Figure 4.52. Comparison metric behavior.

CHAPTER 5

CONCLUSIONS AND CONTRIBUTIONS

5.1 Conclusions and Contributions

The object-based hybrid modeling scheme presented in this thesis provides a useful framework for modeling and analyzing performance of manufacturing systems. The Petri net structures associated with the supervisor and subsystems describe the event-driven dynamics and differential equations describe the time-driven dynamics. Therefore, Petri net analysis techniques such as detecting deadlock or reachability can be applied to the model. In addition, the functional submodel is contained within the Petri net structure, yielding the order of operations. The supervisory Petri net directs unfinished products to destinations through several subsystems.

The mode identification architecture provides a robust method to determine the operating mode of the system by combining both information of events and time-driven dynamics. It is important that the operating mode be detected and used in conjunction with diagnostic and prognostic algorithms in order to relieve the burden of computation and reduce the number of failure detection errors. It is the view of some diagnostic and prognostic developers that the operating mode can be detected via examination of the control inputs or user commands, and thus, not needed. This is true in many cases, however, if the control inputs are affected by failures, then detecting the operating mode in this fashion can give erroneous results. Therefore, utilizing sensor information from the system to detect the mode yields more robust results. However, the mode identification method described in this thesis has its own problems such as how do we select features that are mutually exclusive for operating mode detection and failure

detection. Future work must be done to find ways to derive features in a more automated fashion for many types of system (including ones that contain a great deal of uncertainty), and the types of mode identification modules used in the fusion process.

The reconfiguration methodology that focuses on product defects compares template and on-line feature histograms and a neuro-fuzzy architecture to determine set-point adjustment factors to correct defects. The histograms provide statistical information about the product over several product runs and are typically used in quality control. However, the reconfiguration method in this thesis compares these histograms to yield information about how to correct the defects by adjusting set-points. There is still much to be done on the subject of mapping these comparisons from histogram comparison to set-point adjustment. Future work would include experimenting with different types of feature selection schemes, low-level controller gain adjustments, modeling of the time-evolution of the defects, and the generated bank of histogram comparison features.

The reconfiguration at the system and subsystem levels utilizes the OHM model. It uses the supervisory Petri net to find optimal routes through the system when failures occur. At the subsystem level, the operations can be designed to provide alternate routes while operating in the presence of failures. The low-level controllers of the time-driven systems utilize an adaptive controller framework that can redesign the controller gains online while the system is under non-catastrophic failures. In addition, the OHM model provides a way to divide a single subsystem into many objects that each has a hybrid dynamical model. This gives the system several operational modes to choose from. Future work on this subject is to formally represent the ideas of controllability,

observability, reachability, and stability for the model. In addition, it would be desirable for subsystem operations to organize themselves in the event of a failure.

This research has led to the following main contributions:

1. A new object-based modeling method called the Object-based Hybrid Modeling was developed.
2. A new mode identification method incorporating both events and dynamics information was created.
3. A control reconfiguration methodology was created at the product level to adjust set-points in the presence of product defects.
4. A control reconfiguration strategy was developed at the system level for rerouting of parts through a manufacturing system, rerouting operations, and a low-level adaptive controller structure was suggested for subsystems.

REFERENCES

[1] Al-Hasan, S.A, "Intelligent Approaches to Mission Planning and Control for Autonomous Vehicles," Ph.D. Thesis, Georgia Institute of Technology, 1999.

[2] Allam, M. and Alla, H., "Modeling and simulation of an electronic component manufacturing system using hybrid Petri nets," IEEE Transactions on Semiconductor Manufacturing, vol. 11, no. 3, August 1998.

[3] Andreu, D., Pascal, J-C, and Valette, R., "Fuzzy Petri net-based programmable logic controller," IEEE Transactions on Systems, Man, and Cybernetics, vol. 27, no.; 6, December 1997.

[4] Babaali, M. and Egerstedt, M., "Observability of switched linear systems," to appear in Hybrid Systems: Computation and Control 2004, Lecture Notes in Computer Science, Springer-Verlag.

[5] Balluchi, A., Benvenuti, L., Benedetto, M., and Sangiovanni-Vincentelli, A., "Observability for hybrid systems," Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii, December 2003.

[6] Bemporad, A., Ferrari-Trecate, G., and Morari, M., "Observability and controllability of piecewise affine and hybrid systems," IEEE Transactions on Automatic Control, Vol. 45, No. 10, October 2000.

[7] Branicky, M.S., Borkar, V.S., and Mitter, S.K., "A unified framework for hybrid control: model and optimal control theory," IEEE Transactions on Automatic Control, vol. 43, no. 1, pp. 31-45, January, 1998.

[8] Branicky, M.S., "Stability of switched and hybrid systems," Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, FL, December, 1994.

[9] Branicky, M.S., "Stability of hybrid systems: state of the art," Proceedings of the 36th Conference on Decision and Control, San Diego, California, USA, December 1997.

[10] Branicky, M.S., "Hybrid systems: solutions, stability, control," Proceedings of the American Control Conference, Chicago, Illinois, June 2000.

[11] Branicky, M.S., "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," IEEE Transactions on Automatic Control, vol. 43, no. 4, April 1998.

[12] Branicky, M.S. and Mitter, S.K., "Algorithms for optimal hybrid control," Proceedings of the 34th Conference on Decision & Control, New Orleans, LA, December, 1995.

[13] Branicky, M.S., "Issues and subtleties in hybrid control systems," Proceedings of the 1996 IEEE International Symposium on Intelligent Control, Dearborn, MI, September 15-18, 1996.

[14] Carpanzano, E., Ferrarini, L., and Maffezzoni, C., "An object-oriented model for hybrid control systems," International Symposium on Aided Control System Design, Kohala Coast-Island of Hawai'i, Hawai'i, USA, August 22-27, 1999.

[15] Cassandras, C.G., and Wardi, Y., "Optimal control of a class of hybrid systems," IEEE Transactions on Automatic Control, vol. 46, no. 3, March 2001.

[16] Cassandras, C.G., Pepne, D.L., and Wardi, Y., "Generalized gradient algorithms for hybrid system models of manufacturing systems," Proceedings of the 37th IEEE Conference on Decision & Control, pp. 2627-2632, Dec. 1998.

[17] Cassandras, C.G., and Pepne, D.L., and Wardi, Y., "Optimal control of systems with time-driven and event-driven dynamics," Proceedings of the 37th Conference on Decision and Control, pp. 7-12, Dec. 1998.

[18] Chen, W., Chen, M., Chen, P., Yang, G., and Kang, I., "Fault tolerant control for redundant robot based on joint torque redistribution," 5th Int. Conf. Control, Automation, Robotics and Vision, Singapore, pp. 1325-1329, 1998.

[19] Cheng, G., "Genetic Algorithms and Engineering Design," John Wiley & Sons, Inc., 1997.

[20] Clements, N.S., "Fault Tolerant Control of Complex Dynamical Systems," Ph.D. Thesis, Georgia Institute of Technology, 2003.

[21] Corke, P.I., "A computer tool for simulation and analysis: the robotics toolbox for MATLAB," Proc. National Conf. Australian Robot Association, pp. 319-330, Melbourne, July 1995.

[22] DeGroot, M.H., "Probability and Statistics," Addison-Wesley, 1986.

[23] Deshpande, A. and Pravin, V., "Viable control of hybrid systems," Proceedings of the 35th Conference on Decision and Control, Kobe, Japan, December, 1996.

[24] Demongodin, I., and Koussoulas, N.T., "Modeling of hybrid control systems using Petri nets," Proceedings of the 3rd International Conference on Automatisation des Processus Mixtes: les Systemes Dynamiques Hybrides, Reims, France, March 1998.

[25] Egerstedt, M., Wardi, Y., and Delmotte, F., "Optimal control of switching times in switched dynamical systems," IEEE Conference on Decision and Control, Maui, Hawaii, Dec. 2003.

[26] Egerstedt, M., and Hu, X., "A hybrid control approach to action coordination for mobile robots," Automatica, vol. 38, pp. 125-130, 2002.

[27] Egerstedt, M., Wardi, Y., and Axelsson, H., "Optimal control of switching times in hybrid systems," MMAR'2003, Miedzyzdroje Poland, Aug. 2003.

[28] Groom, K.N., Maciejewski, A.A., and Balakrishnan, V., "Real-time failure tolerant control of kinematically redundant manipulators," IEEE Transactions on Robotics and Automation, vol. 15, no. 6, pp. 1109-1116, December, 1999.

[29] Guler, M., Clements, S., Wills, L. Heck, B., and Vachtsevanos, G., "Generic transition management for reconfigurable hybrid control systems," Proceedings of the American Control Conference, Arlington, VA, June 25-27, 2001.

[30] Hespanha, J., Liberzon, D., and Sontag, E., "Nonlinear observability and an invariance principle for switched systems," Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, Nevada, December, 2002.

[31] Hwang, I., Balakrishnan, H., and Tomlin, C., "Observability criteria and estimator design for stochastic linear hybrid systems," IEE European Control Conference, Cambridge, UK, September 2003.

[32] Ioannou, P.A. and Sun, J., "Robust Adaptive Control," Prentice Hall, 1996.

[33] Johansson, K.H., Lygeros, J., Zhang, J., and Sastry, S., "Hybrid automata: a formal paradigm for heterogeneous modeling," Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System Design, Anchorage, Alaska, USA, September 25-27, 2000.

[34] Juloski, A., Heemels, W., and Weiland, S., "Observer design for a class of piece-wise affine systems," Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, Nevada, December 2002.

[35] Kerrigan, E.C., Bemporad, A., Mignone, D., Morari, M., and Maciejowski, J.M., "Multi-objective prioritization and reconfiguration for the control of constrained hybrid systems," Proceedings of the American Control Conference, Chicago, Illinois, pp. 1694-1698, June, 2000.

[36] Koutsoukos, X.D., Antsaklis, P.J., Stiver, J.A., and Lemmon, M.D., "Supervisory control of hybrid systems," Proceedings of the IEEE, vol. 88, no. 7, July, 2000.

[37] Koutsoukos, X., Zhao, F., Haussecker, H., Reich, J., and Cheung, P., “Fault modeling for monitoring and diagnosis of sensor-rich hybrid systems,” Xerox Palo Alto Research Center Technical Report, March, 2001.

[38] Lygeros, J., Godbole, D.N., and Sastry, S., “Verified hybrid controllers for automated vehicles,” IEEE Transactions on Automatic Control, vol. 43, no. 4, April, 1998.

[39] Lygeros, J., Tomlin, C., and Sastry, S., "On controller synthesis for nonlinear hybrid systems," Proceedings of the 37th IEEE Conference on Decision and Control, Tampa, Florida, December, 1998.

[40] Mahmood, W., “Intelligent Modeling for Control, Reconfiguration and Optimization of Discrete Event Systems,” Ph.D. Thesis, Georgia Institute of Technology, 1997.

[41] Mosterman, P. and Biswas, G., "Building hybrid automata of complex physical systems for real-time applications," Proceedings of the 38th Conference on Decision and Control, Phoenix, AZ, December, 1999.

[42] Narasimhan, S., Zhao, F., Biswas, G., and Hung, E., “Fault isolation in hybrid systems combining model based diagnosis and signal processing,” IFAC, 2000.

[43] Nise, N., "Control systems engineering," The Benjamin/Cummings Publishing Company, Inc., 1995.

[44] Parisini, T. and Sacone, S., “Fault diagnosis and controller re-configuration: an hybrid approach,” Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference, Gaithersburg, MD, September 14-17, 1998.

[45] Pepyne, D.L., and Cassandras, C.G., “Optimal control of hybrid systems in manufacturing,” Proceedings of the IEEE, vol. 88, no. 7, pp. 1108-1123, July 2000.

[46] Propes, N., Lee, S., Zhang, G., Barlas, I., Zhao, Y., Vachsevanos, G., Thakker, A., and Galie, T., "A real-time architecture for prognostic enhancements to diagnostic systems," MARCON, Knoxville, Tennessee, May 6-8, 2002.

[47] Propes, N. and Vachtsevanos, G., "Fuzzy Petri net based mode identification algorithm for fault diagnosis of complex systems," System Diagnosis and Prognosis: Security and Condition Monitoring Issues III, SPIE's 17th Annual AeroSense Symposium, Orlando, Florida, pp. 44-53, April 21, 2003.

[48] Propes, N., Vachtsevanos, G., and Wardi, Y., "An object-oriented hybrid modeling framework for manufacturing systems," IASTED Applied Simulation and Modeling Conference, Crete, Greece, July, 2002.

[49] Ramani, V., “Reconfigurable Control Using Polynomial Neural Networks,” Ph.D. Thesis, Georgia Institute of Technology, 1996.

[50] Sampath, M., “A hybrid approach to failure diagnosis of industrial systems," Proceedings of the American Control Conference, Arlington, VA, June 25-27, 2001.

[51] Slotine, J.E. and Li, W. "Applied Nonlinear Control," Prentice Hall, 1991.

[52] Tomlin, C., Pappas, G.J., and Sastry, S., “Noncooperative conflict resolution,” Proceedings of the 36th Conference on Decision and Control, pp. 1816-1821, December, 1997.

[53] Treister, R. and Herzenberg, W., "Probability binning comparison: a metric for quantitating univariate distribution differences," Cytometry, , no. 45, September, 2001.

[54] Tsuda, K., Mignone, D., Ferrari-Trecate, G., and Morari, M., “Reconfiguration strategies for hybrid systems,” Proceedings of the American Control Conference, Arlington, VA, June 25-27, 2001.

[55] Wang, P., Propes, N., Khiripet, N., Li, Y., and Vachtsevanos, G., “An integrated approach to machine fault diagnosis,” IEEE Annual Textile, Fiber and Film Industry Technical Conference, Atlanta, Georgia, 1999.

[56] Yang, Zhenyu, "A hybrid systems approach towards redundant fault-tolerant control systems," Proceedings of the 39th IEEE Conference on Decision and Control, Sydney, Australia, December, 2000.

[57] Yang, Z. and Blanke, M., “The robust control mixer module method for control reconfiguration,” Proceedings of the American Control Conference, Chicago, Illinois, June, 2000.

[58] Zhang, W., Branicky, M., and Phillips, S., "Stability of networked control systems," IEEE Control Systems Magazine, February, 2001.

[59] Zhao, F., Koutsoukos, X., Haussecker, H., Reich, J., and Cheung, P., “Distributed monitoring of hybrid systems: a model-directed approach,” Proceedings of IJCAI, 2001.

[60] Zhenyu, Y., Huazhang, S., and Zongji, C., “The frequency-domain heterogeneous control mixer module method for control reconfiguration,” Proceedings of the 1999 IEEE International Conference on Control Applications, Kohala Coast-Island of Hawai’I, Hawai’I, USA, August 22-27, 1999.

VITA

Nicholas Chung Propes was born in Milwaukee, Wisconsin on December 13, 1972. He received his Bachelor of Science in Electrical Engineering at Tulane University in 1995, his Masters of Science in Electrical Engineering at The University of Alabama-Tuscaloosa in 1997, and Ph.D. in Electrical Engineering at the Georgia Institute of Technology in 2004. His immediate family resides in Portales, New Mexico. His research interests are hybrid systems modeling and control, Internet control and monitoring, diagnostics and prognostics, image processing, and data mining for Internet search engines. His career focus is to become a faculty member at a university. Nicholas' hobbies are playing the violin and piano, fishing, and writing computer games.